Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Neural Audio Generation for Speech Synthesis

Bachelor's Thesis

In partial fulfilment of the requirements for the degree in Audiovisual Systems Engineering

Author: Georgina Dorca Saez
Advisor: Antonio Bonafonte

Universitat Politècnica de Catalunya (UPC)
January 2018

Abstract

Recently, neural networks have become the state of the art for speech synthesis tasks and they are actually representing a powerful force in the industry.

In this project, we present an implementation of a Text-to-Speech (TTS) system based on deep learning developed in the VEU research lab at UPC, using the Pytorch deep learning library.

For that purpose we adapt two existing systems. The first one MUSA, was a system that predicts the duration of a phoneme and from this generates the necessary acoustic parameters, which were converted to voice by a vocoder. The second one, SampleRNN generated voice in an unconditional way. In this project we have adapted SampleRNN to accept as a conditioner voice parameters; it has been proven on both parameters, extracted directly from the speech as well as with the original predictions of MUSA. Finally, a system that includes both systems in a single neuronal network is implemented, allowing a joint training of the system, which takes the linguistic information from its input and generates speech samples.

On the other hand, some alternatives have been implemented in the representation of the signal, concretely the μ-law quantification has been successfully proposed. Also an implementation of a weight normalization on all the linear layers is performed, improving the original results.

In the first place, SampleRNN was conditioned with the characteristics extracted directly from voice whith Ahocoder, obtaining a signal of high quality and naturalness. Thus the neural vocoder emulates the original Ahocoder decoder with remarkable quality.

Unforgivably, the original MUSA system used has a problem and the reference quality is very poor. Even so, the integrated system has managed to significantly improve the result of MUSA Baseline. In a perceptual test, the 63,8 % of the people consulted have preferred our trained jointly system in front of the independently trained system (with a 7,2%) or on the original baseline MUSA system (29 %).

Resum

Recentment, les xarxes neuronals s'han convertit en l'estat de l'art per a les tasques de síntesis de la parla i actualment representen una força poderosa en la indústria.

En aquest projecte presentem la realització d'un sistema de Síntesis de Veu basat en l'Aprenentatge Profund desenvolupat en el grup VEU de la UPC, utilitzant la llibreria de deep learning Pytorch.

Per tal de realitzar aquesta implementació adaptem dos sistemes existents. El primer, MUSA, era un sistema que predeia la durada d'un fonema i a partir d'aquest generava els paràmetres acústics necessaris, que posteriorment són convertits a veu per un vocoder. El segon, SampleRNN, genera veu de manera incondicional. En aquest projecte, hem adaptat SampleRNN per acceptar com a condicionant paràmetres de veu; ha estat provat en amb tots dos paràmetres, els extrets directament del senyal, així com amb les prediccions originals de MUSA. Finalment, s'ha construït un sistema que inclou tots dos en una única xarxa neuronal, permetent un entrenament conjunt del sistema, que pren a la seva entrada la informació llingüística i genera a la sortida mostres de la parla.

D'altra banda, s'han implementat algunes alternatives en la representació del senyal, concretament s'ha proposat amb èxit la quantificació *mu - law*. També es realitza una implementació de normalització de pesos en totes les capes lineals de la xarxa, millorant els resultats originals.

En primer lloc, condicionem SampleRNN amb les característiques extretes directament de la veu amb Ahocoder, obtenint un senyal d'alta qualitat i naturalitat. És a dir, el vocoder neuronal emula el decodificador Ahocoder original amb una qualitat notable.

Desafortunadament, el sistema MUSA original utilitzat té un problema i la qualitat de referència és molt pobra. Tot i així, el sistema integrat ha aconseguit millorar significativament el resultat de MUSA (Baseline). En una prova perceptual, el 63,8 % de les persones que van ser consultades han preferit el nostre sistema entrenat conjuntament, enfront del sistema entrenat independentment (amb un 7,2 %) o en el sistema MUSA original de referència (29 %).

Resumen

Recientemente, las redes neuronales se han convertido en el estado del arte para las tareas de síntesis del habla y representan una fuerza poderosa en la industria.

En este proyecto, presentamos una implementación de un sistema de conversión texto a voz (TTS) basado en el aprendizaje profundo (Deep learning), desarrollado en el laboratorio de investigación VEU en la UPC, utilizando la libreria de aprendizaje profundo Pytorch.

Con tal de realizar dicha implementación adaptamos dos sistemas existentes. El primero, MUSA, era un sistema que predecia la duración de un fonema y a partir de este generaba los parámetros acústicos necesarios, que posteriormente son convertidos a voz por un vocoder. El segundo, SampleRNN genera voz de manera incondicional. En este proyecto, hemos adaptado SampleRNN para aceptar como condicionante parámetros de voz; ha sido probado en con ambos parámetros, los extraídos directamente del señal, así como con las predicciones originales de MUSA. Finalmente, se ha construido un sistema que incluye ambos sistemas en una única red neuronal, permitiendo un entrenamiento conjunto del sistema, que toma a su entrada la información linguística y genera a la salida muestras del habla.

Por otro lado, se han implementado algunas alternativas en la representación del señal, concretamente se ha propuesto con éxito la cuantificación mu - law. También se realiza una implementación de normalización de pesos en todas las capas lineales de la red, mejorando los resultados originales.

En primer lugar, condicionamos SampleRNN con las características extraídas directamente de la voz con Ahocoder, obteniendo una señal de alta calidad y naturalidad. Es decir, el vocoder neuronal emula el decodificador Ahocoder original con una calidad notable.

Desafortunadamente, el sistema MUSA original utilizado tiene un problema y la calidad de referencia es muy pobre. Aun así, el sistema integrado ha logrado mejorar significativamente el resultado de MUSA (Baseline). En una prueba perceptual, el 63,8 % de las personas que fueron consultadas han preferido nuestro sistema entrenado conjuntamente frente al sistema entrenado independiente (con un 7,2 %) o en el sistema MUSA original de referencia (29 %).

Acknowledgements

First of all, I would like to thank my project advisor for or teaching me everything I know about speech synthesis and enabling me to do this project. Thanks to the help and knowledge that he has given me, it has been possible to carry out this project. I would also like to thank Santiago Pascual for letting me use his project and explain to me whenever I need concepts from both Deep Learning and programming.

Secondly, thanks to the VEU research lab and the people in charge of the computing resources of the D5 building for giving me a place to work in their servers.

Also, I would like thank to all those people who kindly helped in the development of the experiments by taking our subjective evaluations.

Finally, I want to thank my family, along with my friends and my partner for their encouragement all through this project.

Revision history and approval record

Revision	Date	Purpose
0	26/12/2017	Document creation
1	16/01/2018	Document revision
2	22/01/2018	Document revision
3	25/01/2018	Document approval

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Georgina Dorca Saéz	georgina.dorca@gmail.com
Antonio Bonafonte	antonio.bonafonte@upc.edu

Written by:		Reviewed and approved by:	
Date	20/01/2018	**Date**	25/01/2018
Name	Georgina Dorca	**Name**	Antonio Bonafonte
Position	Project Author	**Position**	Project Supervisor

Contents

List of Figures

List of Tables

List of Abbreviations

ADAM	Adaptative Moment Estimation
BP	Back-Propagation
BPTT	Back-Propagation Through Time
CNN	Convolutional Deep Neural Networks
DNN	Deep Neural Network
GM	Gaussian Mixture Model
GPU	Graphics Processing Units
GRU	Gated Recurrent Units
HMM	Hidden Markov Mode
LTSM	Long Short-Term Memory
MFCC	Mel-Frequency Cepstrum Coefficients
ML	Maximum Likelihood
MLP	Multilayer Perceptron
MSE	Mean Square Error
MVF	Maximum Voiced Frequency
NLL	Negative Log Likelihood
NN	Neural Network
pdf	Probability Distribution Function
QRNN	Quasi-Recurrent Neural Network
RNN	Recurrent Neural Network
SPSS	Statistical parametric speech synthesis
TTS	Text To Speech

Chapter 1

Introduction

In this chapter, we present the statement of purpose of this project which provides a general overview of the project. Then, its requirements and specifications are detailed. Finally, we mention the work planning, showing the general project's organization and deadlines, the incidents we have encountered and how they have modified the initial plan.

1.1 Statement of purpose

Recently, end-to-end neural networks have become the state of the art for speech recognition tasks and they are actually representing a powerful force in the industry. This has led to the creation of systems to do the opposite task, generate speech synthesis from raw text. Lately TTS neural systems have become very competitive in front of conventional systems achieving high naturalness scores. The malleability and relative simplicity of these models is compelling, and thanks that nowadays computing power has been greatly enhanced by Graphics Processing Units (GPU), it is possible to work with Deep Neural Networks with reasonably short training times.

Along this project, we present a complete-system deep learning-based TTS system, able to generate a voice signal from characters using two stage RNN based system. With this system we aim to achieve similar naturalness to that offered by conventional systems, but with the flexibility and robustness offered by the neural model. In order to fulfill this task we adapt MUSA (Pascual de la Puente, 2016) for the prediction of vocoder parameters from text, to "condition" SampleRNN (Mehri et al., 2016) neural vocoder, using the TC-STAR dataset 3.3). SampleRNN is an autoregresive generative model based on predicting one audio sample at a given time, using autoregressive multilayer perceptrons, and stateful recurrent neural networks in a hierarchical structure. Then the parametritzation of MUSA will be used in order to condition the generated samples so that at the end the speech follows the input text.

It is ought to mention that along the work we will use various abbreviations in order to entertain the explanations. a list with all the used ones can be found at the beginning of the document.

This project has been carried out at the Signal Theory and Communications Department (TSC) of the Universitat Politècnica de Catalunya (UPC). It has been developed at the Speech Processing investigation group (VEU) Universitat Politècnica de Catalunya as a contribution to its national research project **DeepVoice: Deep Learning Technologies for Speech and Audio Processing**

1.2 Requirements and specifications

The main requirement of this project is to develop a two-stage TTS system based on Deep Learning that generates speech synthesis. The first stage maps the text into acoustic characteristics, which will be used to condition the second stage to generate the audio wave.

For this we must learn the fundamentals and practice of deep learning in order to propose and train a Deep learning Architecture able of generating voice and trace strategies to improve the performance of the model.

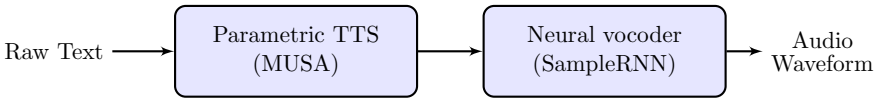


Figure 1.1: Block diagram of a TTS system

The specifications have been decided during the course of the project, taking into account the needs of the system, the resources availability and the proper of the system, and they will be explained along the work.

1.3 Methods and procedures

As will be explained throughout the work, the system is based on two stages (see Figure 1.1). The first one, carried out by MUSA, performs a mapping of the input raw text to its acoustic characteristics. It was implemented as a part of his MSc thesis by Santiago Pascual, who facilitated the code for the realization of this first stage. The next stage is performed by SampleRNN, which constitutes the majority of the work. The code of the baseline model, Uncoditional SampleRNN, is open source thanks to the contribution of Piotr Kozakowski & Bartosz Michalak who provided a Pytorch implementation.

This project has been developed using Python 3 as the programming language. Also we have use Pytorch which is a deep learning framework that provides Tensors and Dynamic neural networks in Python with strong GPU acceleration. Additionally, some Bash scripting has been used in the vocoding stage.

All developed models have been trained on GPU-accelerated servers from the "VEU" group at the Signal Theory and Communications Department from UPC.

1.4 Work Plan

The following Work Packages have been used in our project, the breakdown of each one, as well as the Gantt diagram showing the dead lines are presented in Appendix A

- WP1: Knowledges
- WP2: Programming skills
- WP3: SampleRNN Development (Unconditioned)
- WP4: Conditioning SampleRNN with Ahocoder
- WP5: Integrate MUSA speech parametrization and SampleRNN neural vocoder
- WP6: Documentation

1.5 Incidents and Modifications

Initially we decide to implement a voice synthesis system based on characters, char2wav (Sotelo et al., 2017). This has two components: a reader and a neural vocoder. Neural vocoder refers to a conditional extension of SampleRNN, conditioned under the vocoder acoustic features generated from text or phonemes with an encoder/decoder based on a standard seq2seq attention.

The Neural vocoder adaptation turned out to be more complex than it seemed; understanding the code, preparing the database and carrying out various experiments took longer than expected. In addition, the complexity of the encoder/decoder was too high because of the Attention Mechanisms. This reason prompted us to change the project and focus on the SampleRNN neural vocoder, and adapting it for voice generation using MUSA, as an alternative to predict vocoder parameters from text. MUSA is a two-step system which first predict the duration of each phoneme and then generate as many vocoder frames as required. Therefore, the attention mechanism is not needed.

Finally, the section on innovations could not be performed, which consisted of trying to implement the multi-speaker option, because the incorporation and training of both stages took more time than expected.

Chapter 2

State of the art

Speech synthesis is the process of converting text into voice signal. Previous to Deep Learning, there were essentially two speech synthesis techniques used in the industry: unit selection and statistical-parametric synthesis (SPSS). This chapter gives a brief introduction to the background of this project, discussing the classical methods of speech synthesis (2.1.1 and 2.1.2) as well as a briew review of the Deep Learning topic, its elements, techniques and terminology (2.2). Finally, a brief mention of the state of the art techniques of deep learning applied in this particular task will be made (2.2.6).

2.1 Speech Synthesis

Speech synthesis is the artificial production of human speech by a computer system. Is widely used for many applications such as personal assistant functions, entertainment or assistive technology for the blind or the deafened and vocally handicapped .

In the next section we will review the most used ones, which are the so called Unit Selection systems and the Statistical Parametric Speech Synthesis. Unit selection synthesis provides the best results in quality and naturalness, as long as a sufficient amount of high-quality is given, and thus it is the most widely used speech synthesis methodology in commercial products. On the other hand, parametric synthesis approach offers more flexibility to change its voice characteristics, and more robustness. However the quality of the generated signal is a critic limitation .

On the other hand, deep learning has gained prominence in the field of speech technology, surpassing conventional techniques, allowing a completely new approach combining the potential to provide high quality speech as unit selection synthesis, and the flexibility and robustness of parametric synthesis.

2.1.1 Concatenative synthesis

As the name suggests concatenative synthesis is based on the concatenation of segments of recorded speech called units. According to size of the stored speech units can be phones, diphones or more rarely words or sentences. A unit selection algorithm finds the sequence of units that match best the sound or word to be synthesised, called target. The selection is performed according to the descriptors of the units, which are characteristics extracted from the source sounds, usually a prosodic feature vector with the pitch and duration of the phonemes. Then the selected units can be transformed to completely match the target specification, and at last concatenated. Nevertheless, having a large database, gives a high probability of founding a matching unit, so the need to applying transformations, which degrade the quality, is reduced.

The database contains all source file references, units, unit descriptors, and the relationships between them. Then the target is specified as a sequence of target units with their desired descriptor characteristics.

The main element is the unit selection algorithm. It contains the intelligence of the concatenative synthesis. Units that match best the given sequence are selected from the database. In order to pick the better match, a target distance function and a concatenation quality function is defined.

The target cost C^t, measures the perceptual similarity between the target unit t_τ and the database unit u_i. Is defined as a sum of p weighted differences between the target elements and database feature vectors descriptor of the selected candidates.

The concatenation cost C^c expresses the discontinuity introduced by concatenating the units u_i and u_k from the database. It is defined as a weighted sum of d descriptor concatenation distance functions (as with the target cost). Therefore the concatenation cost of consecutive units in the database is zero.

Usually both costs functions are pondered by a weight ρ which lets us give more importance to one criterion or the other, depending on our requirements. Therefore, given an input sequence of N segments to concatenate, the units are selected according to the next equation:

$$\hat{u}_1^N = \underset{u_1^N}{\operatorname{argmin}}\, \rho \sum_{\tau=1}^{N} C^t(u_\tau, t_\tau) + (1-\rho) \sum_{\tau=2}^{N} C^c(u_{\tau-1}, u_\tau) \tag{2.1}$$

If we consider the unit database as a state transition network where all the states are connected. The unit selection algorithm searches the minimum costly path that represents the target. To do so, we can use Viterbi algorithm (Viterbi, 1967; Forney, 1973) for finding the most likely sequence of the hidden state, called the Viterbi path. Using the target distance C_t as the state cost, and the concatenation distance C_c as the transition cost, the Viterbi algorithm will minimize the sum of both functions.

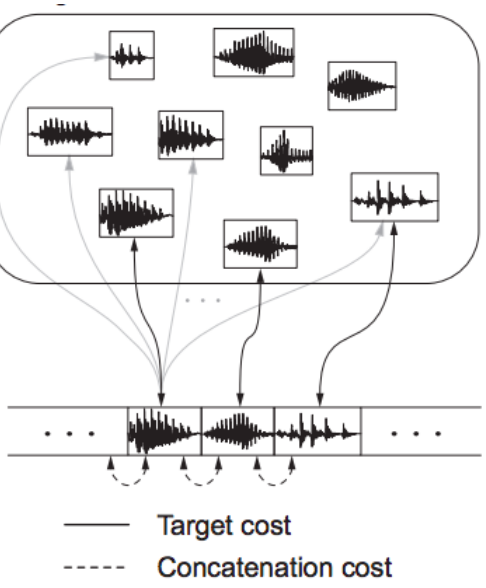


Figure 2.1: Concatenative unit methodology. (Zen et al., 2009)

The final waveform synthesis is done by the transformation and concatenation of the selected units. Possibly a transformation that changes the selected units to match the target more accurately is made, or parameter adaptation such as the pitch. Generally, this technique produces the most natural synthesized speech because is based on real recorded speech. However, sometimes, errors due to discontinuities between two units can be appreciated.

2.1.2 Statistical Parametric Speech Synthesis

The previous technique is very restrictive, because of its large data requirements and development time. Instead of a brute force like method, Statistical Parametric Speech Synthesis (SPSS) models and generates acoustic parameters like fundamental frequency, spectral envelope, etc. which are used to generate speech.

Satistical Parametric Speech Synthesis (Zen et al., 2009), also called hidden Markov model (HMM)-based speech synthesis, uses mathematical models to represent the different sounds and generates speech based on these models. First linguistic features, as phonemes, duration, etc., are extracted after text processing, then are modeled with a generative model, typically HMMs, to represent the corresponding speech signal.

Firstly the conversion of the text into a linguistic specification is achieved, which is known as the "front end". Then extraction of the parametric representations is performed using several processes as a Spectral parameter extractor, to get Mel-Frequency Cepstrum Coefficients, pitch detector, etc.

After the parametrization of each phoneme, a HMM is estimated to model each phone with their respective parametrization features. To do so, a maximum likelihood (ML) criterion is usually used to estimate the model parameters λ. The notation to refer a HMM is $\lambda = (a_{ij}, b_i(y), \pi_i)$; where a_{ij} are the state transition probabilities, $b_i(y)$ the output probability distribution and π_i the initial state probabilities. In our purpose, speech parameters are the observation sequence $O = \{o_0, o_1, ..., o_T\}$ and W are the corresponding set of word sequences. Then we train the acoustic model λ given (O, W) : $\text{argmax}_\lambda \{p(O|W, \lambda)\}$.

To find the optimization that maximizes $P(O|\lambda)$, Baum-Welch iterative algorithm is used until a preset maximum number of iterations is reached, or until there is no improvement between a model λ_t and the next one λ_{t+1}. Once the model λ has been estimated, it can be used to synthesize any state sequence observations, also by means of the ML criterion, finding the most likely trajectories: $\text{argmax}_O \{p(O|\lambda)\}$. These distributions are typically modeled with GMM, which are a linear combination of multiple Gaussian probability distribution functions of the speech parameters vectors.

Although any generative model can be used, historically HMMs have been widely used. Statistical parametric speech synthesis with HMMs is particularly well known as HMM-based speech synthesis (Yoshimura et al., 1999).

Parametrically synthesized speech is highly modular, and flexible. If we can make approximations of the parameters that make the speech, then we can train a model to generate all kinds of speech. And making such a system requires significantly less data and hard work than Concatenative TTS. On the other hand modeling the parameters that make up the TTS, results in a decrease of the quality of the signal, tending to be a flat and monotonous signal.

2.2 Deep Learning

Deep Learning is an area of Machine Learning, composed by a set of tools and techniques that with powerful models can learn complex patterns automatically from data. These methods have drastically improved the state-of-the-art in speech recognition, object recognition, and many other domains such as Speech synthesis.

Conventional machine learning techniques were restricted in their skills to process natural data in their raw form. For many years the construction of a machine-learning system required careful engineering and an expert in the field to design a feature extractor that serialized the raw data (such as the samples of an audio) into a suitable representation from which the learning system could detect or classify patterns in the input (such as acoustic features).

Deep-learning methods are capable of learning multiple levels of representation by composing non-linear modules. Each one transforms the representation of the previous level, or raw input, into a more abstract level representation. Therefore, with the composition of enough of such transformations, very complex functions can be learned. This translates into that

these layers of features are not designed by human engineers, they are learned from data, this represents the big breakthrough of the deep learning, you can let the machine extract the features automatically.

2.2.1 Neural Network

In Deep Learning, the most basic architecture is the so called Neural Network (NN). NNs are defined by a set of basic units called neurons, were a linear operation takes place. Previous to explaining its operating mode, we will explain the basic unit, the neuron.

An input vector $x = \{x_0, x_1, x_2, ..., x_N\}$ is injected into the neuron, then it is ponderated by the set of weights, so that each input link with the neuron has an associated weight. Finally we sum up these products together with a bias term, as seen in the following equation:

$$y = (w^T x + b) \tag{2.2}$$

The result of this linear operation is passed through a function f shown in Equation (2.3) , in order to emulate the biological neurons, which fire an electrical impulse or not depending on a threshold on the input sum. It is usually exemplified with the sigmoid or tahn function.

$$y = f(a) \tag{2.3}$$

A NN is created by hooking together many of those simple neurons, so that the output of a neuron can be the input of another, which is the so called layer, represented in Figure 2.2. The leftmost layer of the network is called the input layer, and the rightmost layer the output layer. The middle layer of nodes is called the hidden layer, because its values are not observed in the training set.

NN systems learn tasks from examples, like people. Imagine that we want to build a system capable of classifying cats, dogs or birds as they appear on the images. We first collect a large data set of images of cats, dogs and birds each labelled with its category. During training, the machine is shown one image and produces an output as a result of the operations, Eq. (2.2) carried out by all neurons, in the form of a vector of scores, one for each category, i.e. a vector with three scores. Then a softmax function is applied in order to squash the outputs of each unit to be between 0 and 1 and divide each output such that the total sum of the outputs is equal to 1. Then the output of the softmax function is equivalent to a categorical probability distribution, i.e tells the probability that any of the classes are true. Thus we want the desired category to have the highest score of all categories, which will mean that it is the most likely category. We compute an objective function that measures the error between the output scores and the desired pattern of scores, which will be a vector with a 1 in the category index, and zeros in the two remaining positions. The machine then modifies its internal adjustable parameters to reduce this error, by the optimitzation of this loss function (see section 2.2.3). These adjustable parameters are the weights of each neuron (see Eq. (2.2)); are real numbers that define the input–output function of the machine.

After training, the performance of the system is measured on a different set of examples called test set. This way we can test the machine ability to produce results on new inputs that they have never been seen during training.

Therefore, the most common architecture is the Deep Neural Network (DNN) which is an stacking of many hidden layers to make the model deeper, i.e. it is a NN with multiple hidden layers between the input and output layers. This hierarchy increases the complexity and abstraction at each level, and it is the key that makes deep learning networks capable of handling very large data sets.

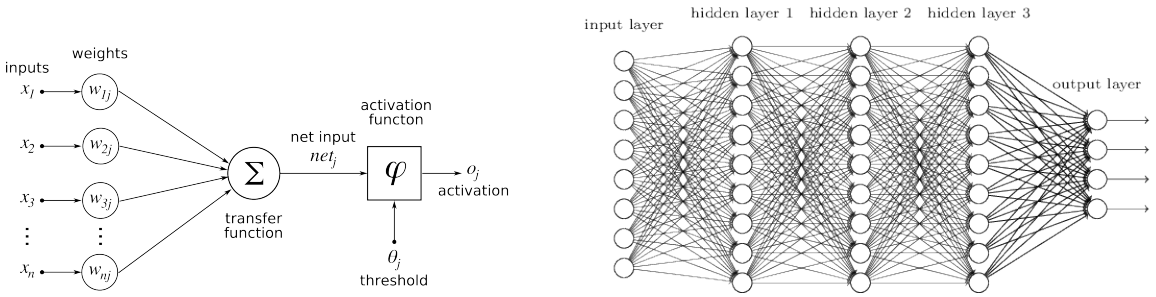


Figure 2.2: On the left we have the basic unit of a neural network, a neuron (a) (Wikipedia, 2005) and on the right a deep neural network (b). (Nielsen, 2015)

Both the NN and the DNN are typically feedforward networks in which data flows from the input layer to the output layer without looping back. But for systems with sequential entries, such as audio or images, there are architectures where the inputs cross the network in different ways, to take advantage of the previous entries, such as Recurrent neural networks (RNNs), which will be explained in the following section, or Convolutional deep neural networks (CNNs).

2.2.2 Recurrent Neural Network

Recurrent Neural Networks (RNN) are a particular type of Neural Network. In a classic NN we assume that all inputs and outputs are independent of each other. For tasks that involve sequential inputs, such as speech, it is often better to use RNNs. RNNs are called recurrent because they perform the same task for each element of a sequence, with the output being depended on the previous computations. In other words they have a "memory" which captures information on what has been calculated previously. They are capable of capturing patterns over time, respecting the ordered context of the sequence. To do so, the outputs of a hidden layer are fed back to the inputs of the same layer, as illustrated in the following figure.

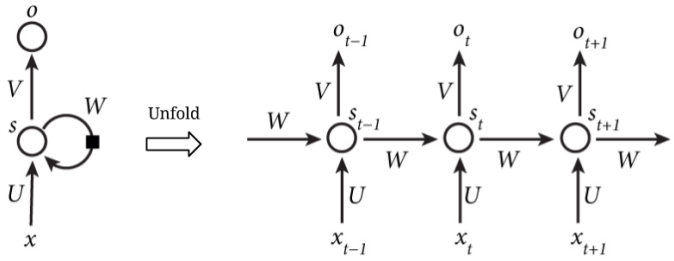


Figure 2.3: A Recurrent Neural Network and the unfolding in time. Its weights are shared across time. (Britz, 2017)

If we check the diagram in Figure 2.3 were we unfold the network for the complete sequence, we can extract the formulas that it uses.

x_t is the input and s_t is the hidden state both at time step t, which is the "memory" of the network. So at a given time t, the RNN has an input vector x_t and the memory state at the previous time s_{t-1}, and generates the new memory state s_t, by the following operation:

$$s_t = g(W \cdot x_t + U \cdot s_{t-1} + b_h) \tag{2.4}$$

W is the weights matrix that goes from input to hidden, this is the feed forward behavior.

U is the weights matrix going from hidden to hidden, where the feedback is made. Also bh is an optional bias vector and f the non-linear transformation, as in Eq.(2.3).

Finally o_t is the output at step t. Which would be a vector of probabilities across our vocabulary.

$$o_t = \text{softmax}(V \cdot s_t + b_v) \tag{2.5}$$

It can be seen it is calculated solely based on the memory at time t.

Unlike a traditional deep neural network, which uses different parameters at each layer, a RNN shares the same parameters (U, V, W shown above) across all steps. This reflects the fact that we are performing the same task at each step, just with different inputs.

There are several types of RNN, the most used are LTSM (Hochreiter and Schmidhuber, 1997) and GRU (Cho et al., 2014) both capture long-term dependencies by adaptively reset or update its memory content. More recently, another type of recurrent unit, to which we refer as a Quasi-Recurrent Neural Network (QRNN), was proposed by Bradbury et al. (2016)

2.2.3 Optimitzation

As stated by Ruder (2015), "deep Learning ultimately is about finding a minimum that generalizes well, with bonus points for finding one fast and reliably".

As explained above, the NN learns from examples, where we teach that each vector of characteristics x represents a category or target t. To perform this learning every time we pass an input through the network we estimate the vector of weights w which fits best the category we want to predict t, this is what we call training. This estimation is done with the gradient descent.

This algorithm minimizes the error function $J(\omega)$, in order that the difference between the output of the network, i.e the prediction, and the target, which is the ideal output, is minimal, which is the ultimate goal of the system. This error or cost function can be defined by the MSE difference, the cross entropy between two probability distributions (Golik et al., 2013), Negative Logarithmic Likelihood, etc.

But in a typical deep learning system, there may be hundreds of millions of these adjustable weights, and also hundreds of millions of labelled examples to train the machine with. In order to adjust the weight vector, the learning algorithm computes a gradient vector in that, indicating by what amount the error would increase or decrease if the weight is modified by a certain amount, called step. This rate at which we go towards the minimum, which depends on the step, is what we call learning rate η. The weight vector is then updated in the opposite direction to the gradient vector, in order to to direct it towards the minimum of the error function, performed like this:

$$\omega = \omega - \eta \bigtriangledown_\omega J(\omega) \tag{2.6}$$

The loss function, averaged over all the training examples, can be seen as a landscape in the high-dimensional space of weight values, as illustrated in the Figure 2.4. The negative gradient vector indicates the direction of most descendent step in the landscape, taking it closer to a minimum, where the output error is low on average.

Before explaining the distressing gradient descent methods, we will explain some necessary concepts. First we define one epoch as one pass of the entire training data set through the network. But as explained below, we do not pass the entire data set into the neural net at once, we divide data set it into batches or sets. Thus we define the batch size as the number of training examples in a pass through the network. Then the number of iterations are the number of passes needed to complete one epoch, each pass using batch size examples. For

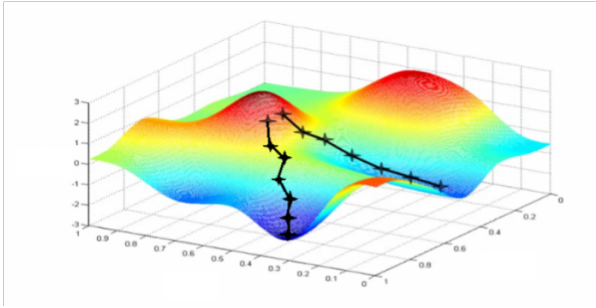


Figure 2.4: Gradient descent path (each black vector represent the movement in one step) over the loss function landscape, averaged over all the training examples. (Ng, 2017)

example if you have 100 training examples, and your batch size is 50, then it will take 2 iterations to complete 1 epoch.

Depending on the used data to calculate the gradient we distinguish two main variants; the stochastic gradient descent and the mini-batch gradient descent. The first one performs the calculation of the gradient updating the loss function for each example, which is to say a batch size of one. This is a fast algorithm but it has much variance in the updates (see Eq.(2.6)). On the other hand, the mini-batch gradient descent performs an update of the loss function for mini-batch of n-examples. With them we managed to reduce the variance in each update maintaining the speed of the algorithm. As a cons the dimension of the mini-batch is one more hyperparameter to keep in mind. This method will be the one used in our project.

There are several gradient descent algorithms, the most famous ones are Momentum, Nesterov, Adagrad, Adadelta, RMSprop and Adam. Adam, Kingma and Ba (2014), will be the one we use in our system. This algorithm moves the weights in a direction based not only on the gradient of the optimization function regarding to the parameters, but also the values of the previous batch gradient. Therefore, the movement of the parameters has a certain inertia or "moment", just as it would happen to a ball when falling to the bottom of the valley. It also adjusts the learning factor specifically for each parameter depending on how the gradients change in the batches: less learning rate the more the gradient changes between batches.

Once explained the algorithm, it will be necessary to explain how the computation of such gradients is made. A fast algorithm known as backpropagation is used. The backpropagation computes the partial derivatives of the loss function $\frac{\partial L(x,w)}{\partial w_i}$ with respect to any weight w or bias b in the network. The expression is somewhat complex, for more detailed information about the mathematics that surrounds see (Hecht-Nielsen, 1989).

The training for RNNs is similar to feed forward ones by means of using a BP algorithm, but here the time dimension also needs to be taken into account. This is important because as mentioned previously, the parameters of the recurrent matrix are shared between time-steps, thus the gradient depends not only on the current time step, but also on the previous ones. For example if we wanted to compute the gradient at t = 5 we would need to back-propagate 4 steps and sum up the gradients. This technique is called Back-Propagation Through Time (BPTT).

Models with a large number of free parameters can describe a wide range of phenomena and can fit any amount of data available. The exclusive use of the training set loss can lead to the phenomenon of overfitting, in which the model fits very well the existing data, that is to say it is specialized on the training data set, but it has a poor performance to predict new results. This means our model does not generalize well from our training data to unseen data.

Among other, dropout layers provide a simple way to avoid overfitting (Srivastava et al., 2014). The idea is to randomly drop a percentage of components of neural network (outputs) from a layer of neural network. This results in that at each layer more neurons are forced to learn the multiple characteristics of the neural network.

2.2.4 Weight normalization

Weight normalization (Salimans and Kingma, 2016), as its name states, consists in normalizing the vectors of weights in any type of neural network architecture, obviating the length of those. Doing so we get a speed up on the convergence of stochastic gradient descent by improving the optimizability of the weights.

Lets consider a standard artificial neural network. Once the computation of each neuron is done (see Equation(2.2) and (2.3)) a loss function is computed to one or more neuron outputs (see 2.2.3), assuming the standard NN is normally trained by stochastic gradient descent in the parameters $\mathbf{w}, \mathbf{b}$ of each neuron. In order to speed up the convergence of that optimization procedure, a reparametrization of each weight vector w in terms of a parameter vector v and a scalar parameter g is purposed (Equation (2.7)).

$$\mathbf{w} = \frac{g}{\|\mathbf{v}\|}\mathbf{v} \tag{2.7}$$

Where g is the norm of the weight vector $||\mathbf{w}||$, and $||\mathbf{v}||$ denotes the Euclidean norm of $\mathbf{v}$, thus the direction of the weight vector is represented by $\mathbf{v}/||\mathbf{v}||$. By doing this we decouple the norm of the weight vector from the direction of the weight vector. Then we perform stochastic gradient descent of the loss function in the new parameters v, g instead ($\nabla_v L$, $\nabla_g L$), improving the convergence of the optimization algorithm.

2.2.5 Embeddings

An embedding is a mapping from discrete values, to dense real value vectors, so instead of representing each discrete value as an arbitrary index integer, they represent it as a vector of real numbers.

NN train best on dense vectors, where all values contribute to define the category. However, many inputs such as words of text, do not have a natural vector representation. Embedding functions transform such discrete input objects into useful continuous vectors.

They are used especially when we have a large level of discrete categories to predict, take for example a 2 million words dictionary data base, were each word is a category, their representation in one-hot vector will be all zeros and a one in the category index, so you would need a 2 million length vector to encode 2 million words. The embeddings makes a mapping transforming the discrete vector in vector of real numbers usually with a lower dimension, providing more information to the system such as similarity in vector space. On the other hand, if the entry is actually a vector of real numbers, normalization is usually applied, to avoid very heavy data distributions at one point but the use of embeddings is not necessary.

In short embedding is basically projecting features to a some higher dimensional space, depending on the task to achieve, so that the features that are more or less alike have a small distance between them in the embedded space. This allows the classifier to learn the representations better and in a more meaning full way.

2.2.6 Deep Neural Network-based Speech Synthesis

Deep Learning has been successfully applied in fields such as Computer Vision, Natural Language Processing and Speech Processing. The recent revival of interest in neural networks has had a strong impact in the area of speech recognition, with breakthrough results obtained by several academics as well as researchers. Therefore the use of Deep Learning techniques in TTS is very recent, next we will explain the top-systems currently used.

Apple's speech synthesis system (Capes et al., 2017), which provides the voices for Siri, use hybrid unit selection, which is similar to classical unit selection techniques, except that they use a parametric approach to predict which units should be selected. The system uses deep learning techniques to predict the target and concatenation reference distributions for respective costs (Equation 2.1) during unit selection. Deep and recurrent mixture density networks can learn the correspondence between text and speech and then map text features into speech features, which are then used to guide the unit selection back-end process.

In the field of statitstical parametric speech synthesis using Deep Leaning, we highlight the work of Zen et al. (2013), who proposes an alternative for the representation of probability densities of speech parameters given texts, based on DNN. With this new architecture solves the limitations of the conventional system, decision tree-clustered context-dependent hidden Markov models (HMMs), which were inefficient to model complex context dependencies. In short, the relationship between input texts and their acoustic realizations is modeled by DNN.

On the other hand there currently exist four main approaches which use neural networks for all or most of the text-to-speech pipeline. The first is WaveNet (van den Oord et al., 2016), a model for audio generation which uses dilated, causal convolutions to form a conditional probability for the next time step value, i.e. a predictive distribution for each audio sample, conditioned under all the previous ones.

$$p(x) = \prod_{t=1}^{T} p(x_t | x_1, ..., x_{t-1}) \tag{2.8}$$

Each audio sample x_t is therefore conditioned on the samples at all previous time-steps.

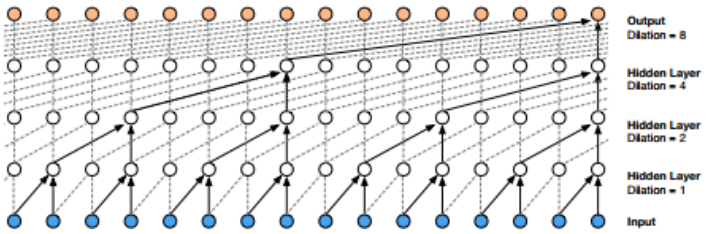


Figure 2.5: Visualization of a stack of dilated causal convolutional layers (van den Oord et al., 2016).

The Figure illustrates how a WaveNet is structured. It is a fully convolutional neural network, where the convolutional layers have various dilation factors.This allows to grow exponentially with depth and cover thousands of time-steps. For TTS tasks, WaveNet is conditioned on linguistic features from an existing TTS system. Given The linguistic feature input l, WaveNets can model the conditional distribution $p(x|l)$ of the audio. In same way, we will carry out the same approach on SampleRNN. Equation (2.8) now becomes:

$$p(x|l) = \prod_{t=1}^{T} p(x_t | x_1, ..., x_{t-1}, l) \tag{2.9}$$

This means it is not fully end-to-end. In addition, its conditional model is auto-regressive and thus prohibitively slow and computationally demanding for many applications. In return for these limitations, WaveNet produces very high quality audio samples, surpassing strong concatenative and parametric baselines in naturalness. Actually is being used to generate voices for the Google Assistant.

DeepVoice from Baidu, implemented the entire TTS pipeline with neural networks (Arik et al., 2017) in contrast to WaveNet. It also achieves production-ready speeds with many stacked QRNN layers which execute in a single batch, greatly reducing compute time. The system comprises five major building blocks: a segmentation model which performs phoneme boundary detection with deep neural networks, a grapheme-to-phoneme conversion model, a phoneme duration prediction model, a fundamental frequency prediction model, and an audio synthesis model, which is a variant of WaveNet. The biggest issue with their approach is that it requires separate training for each block. This greatly increases the complexity of training and deploying their model and makes it harder to adapt existing models to new contexts. Their final naturalness scores (while not perfectly comparable) are significantly lower than WaveNet as a price for the greatly increased speed of their system.

Tacotron is an end-to-end generative text-to-speech model that synthesizes speech directly from characters (Wang et al., 2017). Given `<text,audio>` pairs, with minimal human annotation, the model, based on the sequence-to-sequence (Sutskever et al., 2014) with attention paradigm (Bahdanau et al., 2014), is trained. Unlike previous systems is an integrated end-to-end TTS which is trained altogether, and directly predicts raw spectrogram. Is the only fully end-to-end model. Since Tacotron generates speech at the frame level, i.e. predicts on sample sequences, it's substantially faster than sample-level autoregressive methods, which predicts one sample at a time. The main advantage of this system is that it does not require phoneme-level alignment, so it can easily be trained by using large amounts of acoustic data with transcripts.

Finally Char2Wav is also and end to end approach, but it split its network into two separately trained components; a predictor of vocoder parameters and a neural vocoder (Sotelo et al., 2017). It was our first proposal for this thesis, as explained is section 1.5. It uses a standard seq2seq attention encoder-decoder paradigm for the initial stage and an adapted SampleRNN to compute the final synthesized signal. It also can be trained on characters, however Char2Wav still predicts vocoder parameters before using the SampleRNN (Mehri et al., 2016) neural vocoder, whereas Tacotron directly predicts raw spectrogram.

This chapter has introduced the necessary background for its application in the following chapters. In the next chapter we will explain in detail how the system has been implemented.

Chapter 3

Speech Synthesis using a Neural Vocoder

Along this chapter the different used models as well as the database preparation are specified. Firstly the basic architecture of the Neural Vocoder SampleRNN is detailed, as well as the modifications made to condition it by acoustic parameters (3.1). Then we explain MUSA, in charge to generate the aforementioned acoustic parameters from text (3.2). Is a two-stage architecture; first a duration model predicts the duration of each phoneme, and then the acoustic model generate as many vocoder frames as required. Finally a description of how we structure the input data at the entrance of the different models is detailed (3.3). Besides that, the optimization of the parameters will be ignored, since the generic algorithm of BPTT is used and already explained in section 2.2.3.

3.1 SampleRNN

SampleRNN (Mehri et al., 2016) is an unconditional audio generator based on generating one audio sample at a given time. This model uses two types of networks. In the first thier, we can find memory-less modules, concretely autoregressive Multilayer Perceptrons. In the following tiers stateful Recurrent Neural Networks, in a hierarchical structure (as shown in Figure 3.1) are used. Because its unconditional nature, when training this system with an audio database, such as music, speech, etc., results in a random audio output that maintains the structure of the input.

One of the main difficulties of audio generation is that there is often a very large discrepancy between the dimensionality of the the raw audio signal in contrast to the effective semantic-level signal. Considering the task of speech synthesis, where we typically generate utterances corresponding to full sentences, even at a low sample rate of 16kHz, on average we will have 6,000 samples per word generated. So in order to address the high-dimensionality of raw audio signal it is purposed the use of recurrent neural networks (RNNs) to model the dependencies in audio data, explained in section 2.2.2. An implementation with several RNNs at different scales is used to model longer term dependencies in audio waveforms while training on short sequences which results in memory efficiency during training.

With the purpose of performing a complete-system TTS system we adapt SampleRNN to be able to generate speech, by conditioning the audio data under the vocoder parameters. The next section will explain how it has been carried out. Before getting into the details of the system, it is ought to mention that the author's nomenclature names the input of the system as tier 3, and represents it as the highest level in the structure. According to this, the system's output is named as tier 1. This nomenclature has been maintained, but it results confusing because it is the inverse order of the data path.

3.1.1 Model

As mentioned above, SampleRNN has a hierarchical structure consisting on several tiers. According to the author the optimal distribution, and the one that has been implemented, has 3 tiers. As shown in Figure 3.1, all tiers excluding the last one (tier 1), are the so called frame-level modules, and each one is operating at a different temporal resolution. These are based on RNN, which can be formulated as $\boldsymbol{h_t} = \boldsymbol{\mathcal{H}}(\boldsymbol{h_{t-1}}, \boldsymbol{x_{i=t}})$ with $\mathcal{H}$ being one of the known

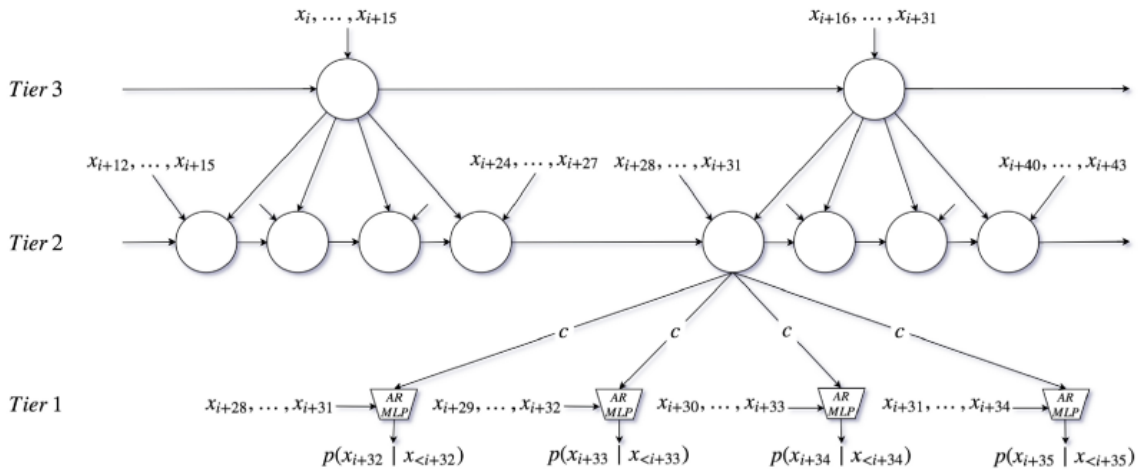


Figure 3.1: View of the unrolled model at timestep i with $K = 3$ tiers, one RNN and up-sampling ratio $r = 4$ for all tiers. (Mehri et al., 2016)

memory cells, Gated Recurrent Units (GRUs), Long Short Term Memory Units (LSTMs), or their deep variations, explained in section 2.2.2. Then the lowest module processes individual samples, which is the so called Sample-level module. This is implemented with a multilayer perceptron (MLP), and the output can be formulated as $Softmax(MLP(h_t))$.

As a result, this architecture allows to model the probability of a sequence of waveform samples $X = \{x_1, x_2, ..., x_T\}$ (a random variable over input data sequences) as the product of the probabilities of each sample under the condition of all previous samples, as illustrated hereunder.

$$p(X) = \prod_{i=t}^{T} p(x_t | x_1, ..., x_{t-1}) \tag{3.1}$$

For our work we use the acoustic features provided by MUSA or Ahocoder, that we are going to name l_i, explained in the following chapter, to condition the previous samples to x_{i+1}. Doing so, we can condition p(X) under the acoustic features l_i, to achieve speech synthesis from $p(X|l_i)$. In the same way as Wavenet (see Section 2.2.6)

$$p(X|l) = \prod_{t=1}^{T} p(x_t | x_1, ..., x_{t-1}, l) \tag{3.2}$$

Correlations exist between neighboring samples as well as between thousands of samples apart, to exploit this information each tier operates on an increasingly longer timescale and a lower temporal resolution. If we look at Figure 3.1 we can observe how at the highest level we have a resolution of 16 samples and 4 at the remaining two. Then each module conditions the module below, which provides the corresponding contextualization of the previous samples. Finally the tier 1 outputs one sample.

3.1.1.1 Frame-Level Module

As we have previously introduced, each Frame-level module is a deep RNN which summarizes its inputs into a conditioning vector for the next downward module. In our adaptation, we will add to the upper tier level a conditioner of the acoustic features that comes from MUSA, as if it where the conditioner of the upper module.

As mentioned above, higher-level modules operate on non-overlapping frames of $FS(k)$ (Frame Size) samples being k, the k_{th} level up in the hierarchy. To ease the explanation we will use the real values of the parameters, $K = 3$ tiers and $FS = [20 \quad 4]$, for level 3 and 2 we will have 80 and 20 samples respectively at the entry of each tier.

We did not choose these parameters randomly, since when conditioning the input samples with the acoustic characteristics it was strictly necessary that the windowing of the input was multiple of the windowing used in extraction of the characteristics. In our case the vocoder/MUSA extracts every 5ms, so that equals to 80 samples. This allows us to condition every 80 input samples under one vector of acoustic characteristics.

Each Frame-Level module is composed of a set of NNs. As shown in figure 3.2 each one with the following structure:

- Input expansion: First of all we can find a 1D convolution[1], on Kernel 1, where we expand the input to the number of neurons in the network in order to feed into the input of the next layer. In our case we used 1024, since it was the author's choice.

$$inp_t = \begin{cases} W_x f_t^{(k)} + c_t^{(k+1)} & 1 < k < K \\ f_t^{(k=K)} & k = K \end{cases} \tag{3.3}$$

- GRU: In second place we find the deep RNN, in this adaptation we use 2 GRU, explained in section 2.2.2, since it was the one that offered the best results.

$$h_t = \mathcal{H}(h_{t-1}, inp_t) \tag{3.4}$$

- Upsampling: Finally we have a last layer where a perform a 1D transposed convolution assigning the Kernel of $FS(k)$. This layer is equivalent to performing an upsampling, by repeating the input into series of r(k) (where r(k) is the ratio between the temporal resolutions of the modules). Since different modules operate at different temporal resolutions we need to make sure that all inputs have the same dimension. For example, as shown in the figure, the input of tier 3 must provide 4 entries of the following GRU, therefore an upsampling of r = 4 will be performed.

$$c^{(k)}_{(t-1)*r+j)} = W_j \, h_t \qquad 1 \leq j \leq r \tag{3.5}$$

The previous structure is shared for all the frame-level modules, but in our adaptation we add a new 1D convolution at the highest tier to expand the vector of acoustic characteristics to the size of the network. This layer will allow us to add the conditioner to the input in the same way that we condition the underlying modules input with the output of the upper tier.

$$inp_t = \begin{cases} W_x f_t^{(k)} + c_t^{(k+1)} & 1 < k < K \\ W_x f_t^{(k=K)} + W_c l_t & k = K \end{cases} \tag{3.6}$$

For our project we have implemented the option to normalize the weights, explained in the section 2.2.4 in order to normalize the weights in each of the layers, if desired.

3.1.1.2 Sample-Level Module

The lowest tier in the hierarchy consists on a Multi Layer Perceptron. It outputs a distribution over the future sample x_{i+1}, by the $FS(1)$ quantified preceding samples, $inp(k = 1)$

[1]Function which multiply a sliding window, called kernel, and then sum them up; we perform it for each element of a 1D vector by sliding the filter over it.

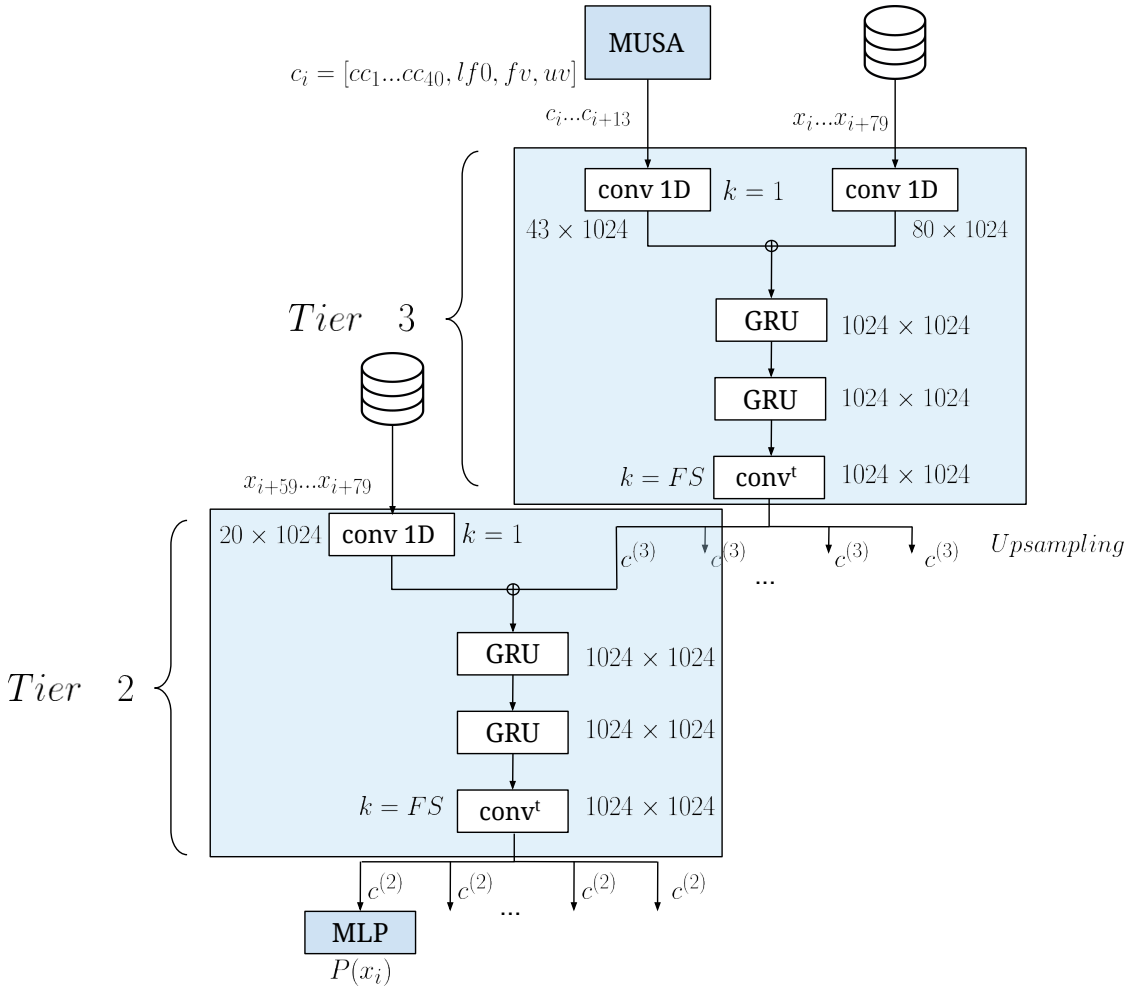


Figure 3.2: Unfolded structure of the first two tiers of the system, with two RNN of 1024 hidden layers, a $FS = [20 \quad 4]$ and an up-sampling ratio of $r = 4$. In the first tier we have an entry of 80 real valued samples and an activated conditioner flag. Both the input and the conditioner pass through their respective 1D convolution layer, and these outputs are linearly added. This will be the entrance to the first GRU, which will store the hidden state $h(k)$ to use it in the next tier as a conditioner. Then the output goes through the transposed convolution, performing the process of upsampling. Finally these output enters to the second frame-level. Instead of adding the acoustic characteristics the input $\text{inp}(k = 2)$ is conditioned under the previous stored memory in the upper tier, $h(k)_{t-1}$ state. From here is followed the same process as the previous tier. Also regardless of the tier, if the input is the first batch that enters the network at any iteration the hidden state will be restarted, since we do not have memory of any previous samples.

conditioned under the upper module output vector $c_i(k = 2)$. As $FS(1)$ is a small value and the correlations to nearby samples are easy to model an RNN is not necessary, instead a MLP is implemented.

Unlike Frame-level module, the previous samples inputs are $\mu-law$ quantified with b = 8 bits. Initially, a uniform quantification was implemented by the author, arguing that a discrete output distribution was mandatory, since the same architecture with a real-valued outputs distribution with GMM generated samples were almost indistinguishable from random noise. In view of the naturalness of the results (see section 4) we add the option to quantify with $\mu - law$, offering also a discrete output but reducing the dynamic range of the audio signal.

The structure of this module, ilustrated in Figure 3.3, is as follows; first of all an embedding step maps each of the q discrete values to a real-valued vector embedding e_i, explained in section 2.2.5.

$$f_i^1 = flatten([e_{i-FS^{(1)}+1}, ..., e_i]) \tag{3.7}$$

Subsequently the MLP, consists of 3 layers. The Input layer is the same as the frame-level one, where we expand with a 1D Convolution the input to have the size of the network and the conditioning of the previous layer, and thus be able to add them.

$$inp_i^{(1)} = W_x^{(1)} f_i^{(1)} + c_i^{(2)} \tag{3.8}$$

The following is the Hidden layer, is shaped of nonlinearly-activating nodes with the same input and output dimension, 1024x1024. Finally the output layer returns q-way discrete distribution over the 256 possible quantified values of xi. Since MLPs are fully connected, each node in one layer connects with a certain weight w_{ij} to every node in the following layer.

$$p(x_{i+1}|x_1, ..., x_i) = Softmax(MLP(inp_i^{(1)})) \tag{3.9}$$

Finally, a Softmax function is applied to the output. Achieving the probability of a sample (3.1).

3.2 Extending MUSA with a Neural Vocoder

The following section will explain MUSA (Pascual de la Puente, 2016), that we used as the first stage of the TTS. This system was the result of the master's thesis of Santiago Pascual, at UPC. MUSA is a two stage RNN-LSTM model, which from contextual features predicts the duration of each phoneme, with duration model; those are sent to the acoustic prediction system which will generate the acoustic parameters for the SampleRNN conditioning. This two stage architecture was influenced by the work in Zen and Sak (2015). Finally the integration of both systems will be carried out, to perform a joint training.

3.2.1 Model

First raw text is inserted by the user and the text analysis front-end of Ogmios (Bonafonte et al., 2006), which is a TTS system developed by UPC including modules for text processing, prosody modeling and speech generation, converts it to contextual features. Then these features are injected into the duration model which predicts the duration for the current phoneme. Then with that duration predicts the acoustic frame coefficients, for as many frames the duration prediction return, also taking the linguistic features as a target. In both

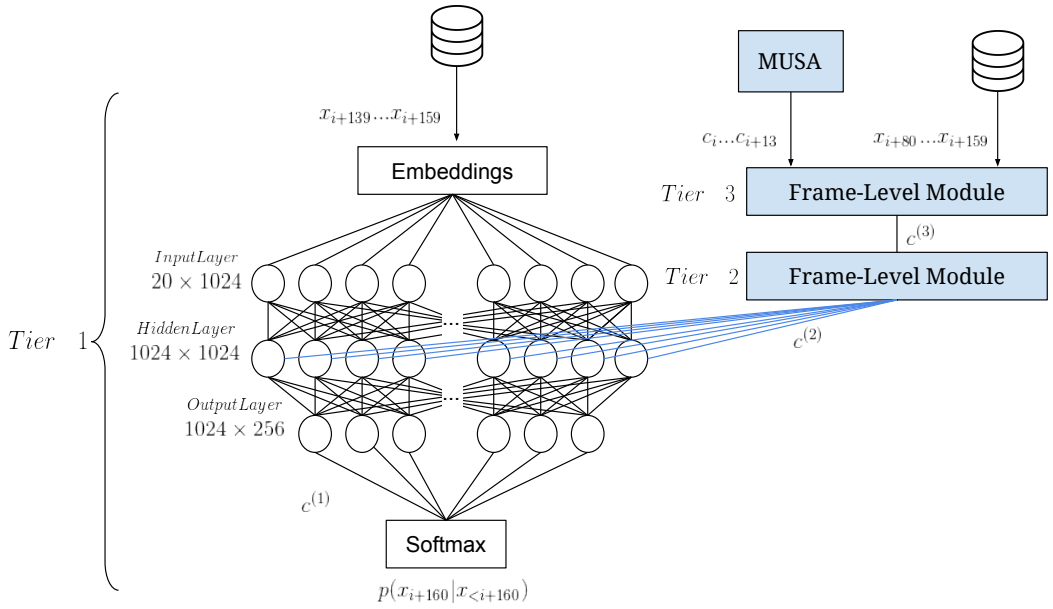


Figure 3.3: Unfolded structure of the final tier of the system, an MLP of three layers, a FS=[20 4] and an up-sampling ratio of r=4.

stages uses LTSM, explained in section 2.2.2, trained both to learn a mapping function from linguistic features (inputs) to acoustic features (outputs).

3.2.1.1 Duration RNN

The duration model is in charge of predicting the prosodic information. It uses a LSTM-RNN architecture, designed to model temporal sequences and their long-term dependencies, concretely a LTSM of one hidden layer with 256 units. Afterwards, the output of the duration model is a linear feed forward layer, also called Fully Connected layer (FC), besides it applies a dropout (see section 2.2.3) of 0.5 between them.

The duration module predicts the real amount of time that the current phoneme lasts from the contextual features. Contextual features are generated by Ogmios from analyzing the textual transcriptions of the database recordings, which will be presented in section 3.3.2. Then the duration is log-normalized and then the max-min (between 0, 1) is applied (Eq. (3.10)). From this it generates a vector with the relative durations (3.11), which are the frame windows that the acoustic model will receive, normalized by the total duration of the phoneme. Moreover, to adapt the input for the acoustic model, the labels of the current phoneme is also added to the output vector. At each acoustic model input vector we will have the label, the relative duration and the total duration of the phoneme, having then contextual information from where is the frame in the phoneme timeline. In short, predicts the amount of acoustic frames to be generated by the acoustic model.

$$\hat{d} = \frac{\ln d - \ln d_{min}}{\ln d_{max} - \ln d_{min}} \tag{3.10}$$

$$\hat{r}_d = \frac{r_d}{d} \tag{3.11}$$

On this model no changes have been made for the training algorithm, but the generation of durations for the synthesis has been edited. In the synthesis duration model generated so many frames, that is to say relative durations, as they were necessary to cover the whole phoneme. This implied that, on occasion, more acoustic vectors would be generated than

samples in the audio. In case that the last samples of the phoneme were less than the frame size, one frame would also be assigned. At the time of entering the conditioning to SampleRNN, the 80 samples were desynchronized with the corresponding acoustic frame, because there were more frames than samples. For this reason, the function of synthesis is reimplemented by displacing the thresholds that delimit the phoneme duration in order that they were multiples of 80, thus fitting exactly the necessary windows in each phoneme duration. Moreover, the maximum error that could be generated when reassigning 80 samples from one phoneme to another is not relevant.

3.2.1.2 Acoustic RNN

Here the acoustic model is explained. This model is also built with Recurrent Neural Networks, but unlike the duration model the output layer is also recurrent. It is also distinguished by making an embedding for first projections of the data with 2 Embedding layers of 256 (see section 2.2.5) . The structure consists of a LTSM of two hidden layer, both of 256 units with a dropout of 0.5 between them.

The first two Embedding layers at the input are Fully Connected layers with tanh activation functions, which are used to make a first mapping of the mixed sets of features that arrives from the duration model. Then they are injected directly to the LSTM hidden layers. Finally the output layer is an LSTM layer with 43 output cells which generates the output vector, corresponding to the acoustic parameter trajectories for the Vocoder.

In order to predict all the acoustic parameters we use as the input the same linguistic inputs, as before, to take into account not only the phoneme, but also the context that surrounds it. In addition, we must take care with the duration that has already been predicted, so that it becomes an input of the acoustic stage with a min-max normalization Eq.(3.10) with also the relative duration expressing what is the current position in the predicted frame duration Eq.(3.11). Both of these duration features are concatenated to the linguistic inputs and then fed into the acoustic model.

Finally we add the target which is the acoustic frame from Ahocoder 3.3.3, normalized with a min-max, in order to compare the predicted output with the desired one.

Once the model is trained a denormalization would be made to recover the acoustic characteristics. In our reimplementation this step is ignored and the normalized input is directly added to SampleRNN, saving us the denormalization in MUSA and a new normalization in SampleRNN. On the other hand, the models code has not been modified, however, in order to adapt both stages, a modification of the data management is made, it will be detailed in the next section.

3.3 Preparation of the Data

In this section the data management for the input of the models discussed previously will be explained. Although it seems a trivial part how the data has been structured in order to make it "understandable" at the entrance of our network it has turned out to be the most difficult part to implement.

3.3.1 Data structure

Since two systems are used for the two stages of the TTS the data management for each one of them will be detailed. As will be explained below Pytorch uses two main functions for data management, *Dataset* and *Dataloader*. The first one stores the data, so when requested by the Dataloader function returns a input vector ready to input the network. Dataloader

groups these vectors in batches to perform a backpropagation with a mini-batch Gradient descent, explained in 2.2.3.

First, the Dataset function has been re-implemented for the Un-condicional SampleRNN, so that any conditioner can be added to the data structure. All the audio files (.wav) of the desired directory are read and quantified, either linearly or μ-law, as explained in section 3.3.4.

Once quantified they are concatenated in a vector, which we store in memory. In this way, there is no need to load the data each time the Dataset is called, as in the author's version. Then a reshape of the vector is performed in order to restructure it in a batch size matrix by a remaining dimension, it will vary according to the number of concatenated samples.

In this way every time we want to generate a batch for a system input, through the call of the Dataloader, we can select by columns, thus maintaining the order of the sequences. This disposition is the so called stateful data. Specifically, we will select fixed lengths of seq_len, typically 1040, on which we will perform truncated backpropagation through time.

For the input we take an overlap, which for the first batch will be a vector of zeros, of $FS(3)$ lenght, i.e. 80 , followed by seq_len samples minus the last, which is the sample to be predicted. For the target then we use the whole seq_len, including the last one that will be the target itself. Finally a flag called $reset$ is added, which we use to warn that there is no memory before this data and it must initialize the hidden states of the RNNs, that is, it will be active only in the first batch where the overlap are zeros. In short, every time Dataset is called returns $(input, target, reset)$ vector.

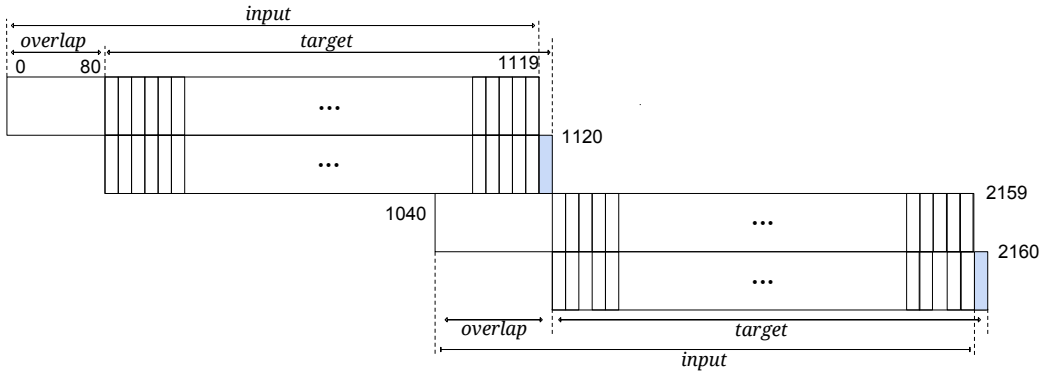
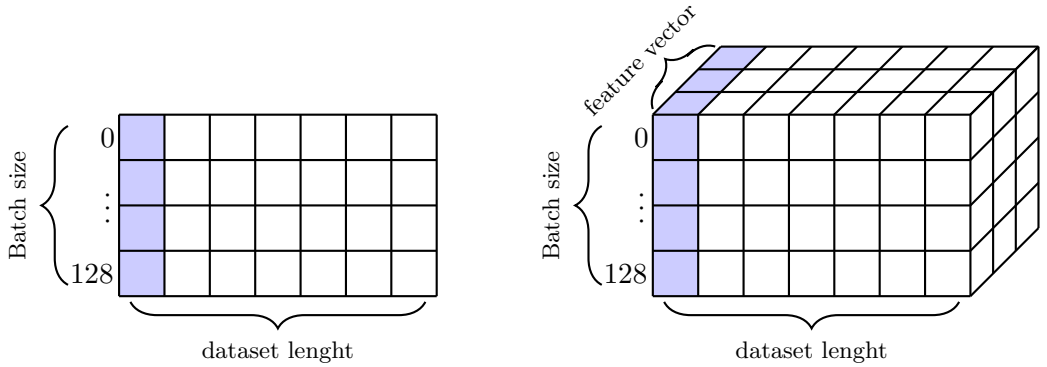


Figure 3.4: Extraction of input and target samples from the data set matrix.

Then the Dataloader, calls the Dataset batch size times, to form the input batch that will entry to the network. As we have explained, the data is stateful, so when the dataloader asks for the first vector of the second batch, it will be the continuation of the first vector of the previous batch, and so on. This allows to exploit the "memory" of the RNN.

For the incorporation of a conditioner we edit the dataset by reading the cepstral coefficients, pitch and maximum voiced frequency generated by the Ahocoder and later on we interpolate them (see section 3.3.3). Then a control is carried out between samples and feature vectors, in order to have all the data synchronized, so that for every 80 samples there is a single vector of characteristics. Since the database is recorded at 16kHz, and Ahocoder extracts the parameters every 5ms window, it means that features are extracted every 80 samples.

In case that the total number of samples per file is not multiple of 80, would mean that the last vector of characteristics does not have its 80 corresponding samples. In that case the samples and the characteristic vector will be discarded if there are less than 60 remaining samples, otherwise a zero padding will be made to cover the missing samples and we will keep the features vector.



(a) Unconditioned SampleRNN Data set structure

(b) Conditioned SampleRNN Data set structure

Figure 3.5: Data set matrix structure, for Uncondicional and Conditioned SampleRNN

From here on we perform the same concatenation process, this time with vectors, so we will have a flat matrix of the same width and length as the samples matrix, with a depth of 43, which is the dimension of a vector of characteristics 3.6. Thus, when Dataloader asks for an input it will receive $(input, target, reset, conditioning)$ vector.

It is important to highlight the use of a *seq_len* multiple of 80 in the conditioned system, in order to add vectors of characteristics to each 80 samples synchronized. The same applies to $FS^{(3)}$, which determines the overlap of the sequence and the length of the upper tier frame, it must be at least 80 to have an associated features vector.

Finally, an adaptation of the MUSA acoustic model Dataset is carried out to do the re-training of both systems jointly. The Dataset function was re-implemented by adding the wav files and synchronizing them with the duration frames, which we have previously called relative durations. Also we eliminate the acoustic vector from Ahocoder, due to the fact that the system enters durations and linguistic features and will generate samples, the acoustic parameters will be evaluated implicitly when comparing the output samples. Thus we have all the frame segments of audio parsed with their corresponding phoneme and relative duration, since the entrance of the acoustic model needs frame size inputs.

The labels are then encoded, as explained in section 3.3.2, and we create frame vectors, which contain the duration, the linguistic features and the samples of the current frame. Subsequently, all frame vectors are concatenated consecutively, in order to repeat the process of arranging the data in an stateful way, as previously explained. Concretely by placing the vectors in a matrix, so that in each row there is *aco_max_seq_len* frame vectors, we use 13 because they are equivalent to *seq_len* samples that we need in SampleRNN. The matrix is arranged so that every "batch size" rows are sequential and so on, as we perform for SampleRNN, i.e the i row is sequential with the $i + 128$ row.

Finally, the same division-targeting process is performed as in SampleRNN Dataset by taking the *seq_len* samples of the 13 frame vectors and adding the overlap with the last frame vector of the previous sequential row, that is, 128 rows above. Obtaining this way for each call from the Dataloader a vector with 13 *code labels*,13 *total durations*, 13 *relative duration*, the *input* with a length of 1119 samples and *target* with a length of 1040 samples.

3.3.2 Text to Label and encoding of the Data

A processing of the raw text into a label representation is generated by the front-end of Ogmios (Bonafonte et al., 2006), as introduced earlier. This representation is composed of a set of contextualized prosodic and phonetic features, which are a phonetic transcription of a

Label Information
{preceding, succeeding} two phonemes
Position of current phoneme in current syllable
Number of phonemes at {preceding, current, succeeding} syllable
{accent, stress} of {preceding, current, succeeding} syllable
Position of current syllable in current word
Number of {preceding, succeeding} {stressed, accented} syllables in phrase
Number of syllables {from previous, to next} {stressed, accented} syllable
Guess at part of speech of {preceding, current, succeeding} word
Number of syllables in {preceding, current, succeeding} word
Position of current word in current phrase
Number of {preceding, succeeding} content words in current phrase
Number of words {from previous, to next} content word
number of syllables in {preceding, current, succeeding} phrase

Table 3.1: Context-dependent label format (Zen et al., 2013).

few windowed phonemes, in this way, we contextualize the current phoneme thus facilitating the co-articulation between them. Also, information about stressed syllables, position of the phoneme inside the current syllable, the position of the syllable in the word, etc. is embedded in these features. Some examples are represented in the following table:

Once the labels are generated, we normalize them in order to proper the learning process because of the non linearities of the network (LeCun et al., 1998). First the distance features in the label are z-normalized, such that it has mean $\mu = 0$ and $\sigma = 1$:

Then the categorical values have to take some numeric type, to that end we use a one-hot code, so that (A,B,C) would be encoded like `(001,010,100)`, where the position of the bit `1` tells us which category do we have. Note that in the case of one-hot codes we have many bits for a symbol, therefore having B bits for a one-hot code means having B inputs in the network. This implies that the codebook generated in the training, where each label is mapped to the one-hot vector, it must be the same in test and validation since the network learns with the said mapping.

3.3.3 Vocoding

The Vocoder takes the raw speech and, by windowing it, it extracts many acoustic frames composed of features that describe the signal in a more convenient way. For that purpose, we have used Ahocoder, an Harmonics Noise Model vocoder, by Erro Eslava (2017) and Erro et al. (2011). Concretely extracts Mel cepstral features (MFCC) that model the spectral envelope, the pitch of the signal in log scale ($\log(f_0)$) and its Maximum Voiced Frequency (MVF).

We dictate Ahodecoder to use a sliding window of 5ms what is translated into 80 samples since our database is recorded at 16kHz, hence generating the following parameters:

- `.mcp` - Mel Frequency Cepstral Coeficients (MFCC) of order $p = 39$ - 40 cepstrum parameters. They represent the short-term power spectrum of a sound.
- `.lf0` - Pitch Contour - 1 lf0 . Fundamental frequency of speech signal (logarithm).
- `.vf` - Maximum voiced frequency (MVF) - It models the spectral frequency separating periodic regularities (harmonics) and aperiodic components.

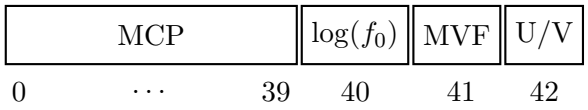


Figure 3.6: Structure and dimensions of a feature vector.

Because the pitch contour signal behaves very different for voiced signals, where it is a continuous signal, than for unvoiced signals, which goes to $-\infty$ we linearly interpolate these parameters to mitigate those effects. But if we use this data at the entrance of the acoustic model, it would not be able to learn the real distribution of the pitch. To solve it we save a flag for each frame indicating if it contains Voiced or Unvoiced speech so the acoustic model mask out the interpolated values when it is an unvoiced frame.

3.3.4 Normalization

The speech parameters we have used have been normalized in order to control the magnitude of the gradients in the training. During the development of this project we have used two different normalizations. The first one we used is a Standard score normalization, and the second is a Feature scaling.

This normalization guarantees that $x'_i \in [0, 1]$, where x_i is the ith dimension of an input parameter vector. Equation (3.12) depicts this scaling:

$$\boldsymbol{x'} = \frac{\boldsymbol{x} - \boldsymbol{x}_{min}}{\boldsymbol{x}_{max} - \boldsymbol{x}_{min}} \tag{3.12}$$

$\boldsymbol{x}_{max}$ is a vector containing the maximum value in the Dataset for each parameter. $\boldsymbol{x}_{min}$ contains the minimum value in the dataset for each parameter. All operations in the equation are element-wise.

3.3.5 Quantization

The author showed that same architecture on real-valued generate samples almost indistinguishable from random noise. This phenomenon is due to the fact that when estimating the pdf during the training, we select the best value as the average of the pdf, to obtain the most probable value. But in generation said average is not usually the most probable value, i.e. the likely sample. When quantifying, we discretize said pdf choosing the highest, therefore more probable, value of this one. For this reason, we perform a quantification, assigning an interval of the input signal to a single level of output. In this work we use quantization with $q = 256$, corresponding to bit depth of 8.

Two different quantification methods are tested in our system: Uniform and μ-law. Te first one was purposed by the author. The distance between the reconstruction levels is always the same. They do not make any assumptions about the nature of the signal to quantify. In contrary, μ-law exploits the characteristics of the voice signals, which are formed largely by small amplitudes, since they are the most important for speech perception, therefore these are very likely. On the other hand, large amplitudes do not appear as much, therefore they have a very low probability of appearance. In this way the levels of quantification are distributed in such a way that a greater number of levels is assigned for the smaller amplitudes (since they contain the most of the density information) and a smaller number to the high ones.

Once we have introduced the structure and functioning of both the data set and the model, we will define, in the following chapter, an experimental framework to validate the model and select the best architecture and optimization method.

Chapter 4

Results

This chapter presents the results obtained with the different systems. First, the experimental methodology is detailed, where we introduce the used Database (4.1.1) as well as the used metrics (4.1.2 and 4.1.3). Then a review of the different results obtained with the uncondicional SampleRNN by making a brief architecture search to tune the different parameters of the network in an objective way is performed. On the basis of that results obtained, the conditioning SampleRNN under the Ahocoder acoustic parameters is evaluated objectively (4.2.2); also a study of the behavior of the system with different optimizers is discussed. From the objective results obtained in the previous systems we have used the best architecture and optimizer, for an objetive (4.2.3) and subjective evaluation (4.3) of the complete system, MUSA and SampleRNN, trained independently (4.2.3.1) and jointly (4.2.3.2).

4.1 Experimental Methodology

4.1.1 Speech database

In this section we introduce the *Spanish TC-STAR TTS Speech Database*, which has been used to train and evaluated the proposed TTS. This speech corpus was produced by UPC durig the European project TC-STAR (2004–2006) and is distributed by ELRA (ELRA-B0014).

The speech data designed for TTS contains speech in Spanish, English and Mandarin. In this project we have used the male voice of the Spanish corpus, which contains approximately 10 hours of speech. Speech samples are stored as sequences of 24-bit 96 kHz but for project purposes downsampled to 16kHz and 8 bits and saved in a WAVE (`.wav`) file. The used training/validation/test split is 80%-10%-10%.

4.1.2 Objective Evaluation of the Architecture and Learning Strategy

All the models have been trained with stochastic gradient decent (section 2.2.3), using mini-batch of size 128, to minimize the Negative Log-Likelihood (NLL) in bits per dimension (i.e., per audio sample).

Likelihoods are conditional probability densities, specified by its pdf $f(x, \sigma)$, where x represent a single observation, concretely a given sequence of samples, and σ the parameters of a statistical model. In our case x is fixed and we use the pdf to compute the likelihood of the parameters σ, as $f(x_n|x_1^{n-1}, \sigma)$, to pick the parameter that most likely gave rise to our data. This expression gives the argmax over a single data point, i.e. training observation, where x_1^{n-1} are the previous samples (input) and x_n the sample to predict, preceded by a small context (target). We can define the joint likelihood of the parameters over all observations by multiplying each likelihood. Thus will return a greater value the more they resemble. But since in the optimization we search for minima not the maxima, a negative of this product is performed, Negative Likelihood.

$$\mathcal{L}(\sigma|x) \equiv f(x|\sigma) = \prod_{n=1}^{N} f(x_n|x_1^{n-1}, \sigma) \tag{4.1}$$

Since the product of numbers between [0,1] gets very small very quickly a logarithm transforms the product of potentially small likelihoods, into the sum of logarithms, and as it is

a growing monotonous function it does not modify the result of the optimization; is the so called Negative Logarithmic Likelihood.

In order to quantify how well our system is performing, we will use the mean of the NLL generated in each iteration during 30 epochs. We use 30 epochs because it is enough to analyze the behavior of the system in all the experiments.

4.1.3 Subjective metrics

Although the previous evaluations will allow us to verify the convergence of the optimization algorithms and compare different aspects, in the synthesis of voice and other applications associated with human perception, the final evaluation requires subjective tests carried out by people. Thus we perform a subjective evaluation, where five random sentences were chosen from the test split of TCSTAR database. We ask some volunteers to rate the selected audio files obtained from the most outstanding architectures, in a scale from one to five, one being bad quality and five being excellent quality.

4.2 Objective results

We particularly explored the response of the system with the different types of quantification explained in section 3.3.4, as well as a tune of the different parameters of the network, including the use or not of the normalization of the layers. On the other hand, in view of the results obtained in the conditioned SampleRNN experiments we studied the response of the system using different optimizers.

4.2.1 Unconditioned SampleRNN

Initially we carried out a study on the unconditional model in order to select the optimal parameters of the architecture that we would use in the final TTS system.

It was analyzed how the use of the different quantification methods, presented in section 3.3.5, affected the output of the system. If we observe the table 4.1 we observe that Uniform quantization offers a lower NLL, but when we synthesized an example we realized that a constant background noise could be appreciated. Instead a μ- law quantization was purposed, generating a cleaner sound.

Due to the controversy that the uniform quantification offered better results in terms of NLL than the μ-law quantification, but at the same time generates a worse synthesis, we decided to study the speech entropy of the quantized signal with both methods.

Just to have a reference of the speech entropy: for 8 bits a zerogram [1]: has an entropy (H) of 8 and a perplexity (PP) of 256.

Using 800 seconds of samples of same TCSTAR data, i.e. 12.8 million samples, we analyze with a language model tool with the same training and test set the resulting entropy after the quantification. As it is illustrated in Table 4.1 the entropy of the data, once quantified, is similar to the NLL averaged obtained results.

In the case that the audio signal would have a likelihood of occurrence of all amplitude levels alike, the ideal quantification would be the uniform, but in the case of speech signals, statistically low amplitude levels appear much more frequently. Therefore the μ - law quantification spreads the data generating thus more entropy in the resulting signal. Contrary to the fact that the uniform quantification does not take into account the mentioned disposition, so

[1] An n-gram is a contiguous sequence of n items from a given sample of text or speech,

Quantitzation	Method	Average NLL Test
$\mu-$law(8 bits)	4-GRAM	3.36
	SampleRNN	2.83
	SampleRNN + Weight normalization	**2.39**
Uniform(8 bits)	4-GRAM	1.43
	SampleRNN	1.55
	SampleRNN + Weight normalization	1.85

Table 4.1: Averaged NLL in Test for different quantification algorithm and the using of Weight Normalitzation

Model	Average NLL Test
3-tier	2.67
2-tier	**2.39**

Table 4.2: Averaged NLL results depending on the number of tiers of Frame-Levels in the system.

it would concentrate more values in the lower levels and surroundings, which are the more probable, creating a more compact distribution of the information and therefore less entropy in the signal.

Obviously, SampleRNN reduces the entropy, as it uses more information and the prediction model is more complex. But just to have an idea of how much is the gain (shown in Table **??**) and we can compare the entropy of the data itself with the SampleRNN generation (shown in Table 4.1).

Furthermore, when we performed the quantitzation tests, we realized that in the NLL loss curves some peaks appeared after certain iterations. The conclusion we draw was that the gradient was triggered, this problem is also known as exploding gradient. As explained by Schoenauer-Sebag et al. (2017) it occurs when the learning system abruptly meets a cliff structure in the gradient landscape (see Figure 2.4), usually due to too large learning rate. Therefore, it was decided to add the Weight Normalization, explained in section 2.2.4. If we look at the Figure B.1 in the appendices it can be appreciated that by using weight normalization we eliminated the mentioned peaks. Therefore, the Normalization Weight has been used for all the linear layers in the model to soften the curves, thus reducing the NLL average.

Finally, a study on some tuneable parameters of the frame-level tiers was made. The Table 4.2 shows how the amount of frame-levels used affects to the averaged NLL in test. As proposed by the author, using 2 tier of frame-levels, the lowest NLL is achieved.

In terms of the amount of hidden layers that compose each of the GRUs we observe that it does not mean much difference in terms of NLL between 512 and 1024. We finally pick 1024 hidden layers because it offers a slight improvement.

Model	Average NLL Test
256 hidden layers	2.44
512 hidden layers	2.33
1024 hidden layers	**2.32**

Table 4.3: Averaged NLL in Test results depending on number of units in each hidden GRU layer

Optimizer	Average NLL Test
Adam	2.56
RMSprop	3.19
Scheduler	**2.41**

Table 4.4: Comparison of the different tested optimizers in terms of NLL.

4.2.2 Conditioned SampleRNN under Ahocoder acoustic features

For the conditioned system under the vector of characteristics extracted from Ahocoder we have used the best architecture previously selected in the Uncondicional model. With that architecture we observed that the output curve made a sudden change at 15th epoch as it shows the Figure 4.1. In order to find the reason, we analyze the behavior with different optimizers. We believe that the learning rate could be too big after that epoch, thus we decide to force the decrease of the learning rate monotonically, with the so called scheduler.

It change the learning rate to the initial lr decayed by gamma every $step_size$ epochs. Concretely we set up an initial $lr = 0.001$ with a decay of gamma 0.1, thus having:

$$\begin{cases} lr = 10^{-3} & \text{epoch} < 15 \\ lr = 10^{-4} & 15 \leq epoch < 35 \\ lr = 10^{-5} & epoch > 35 \end{cases} \tag{4.2}$$

Using the Scheduler, with the Adam Optimizer, we managed to solve the problem. On the other hand we test RMSprop, an optimizer that like Adam, is an adaptive learning rate method, to check if the same effect occurs. As shown in the Appendices B.2 we have a curve with higher mean error, but not generating the abrupt jump in the 15th epoch. We assume that due to having different learning rate adaptation algorithms, does not fall into the same error.

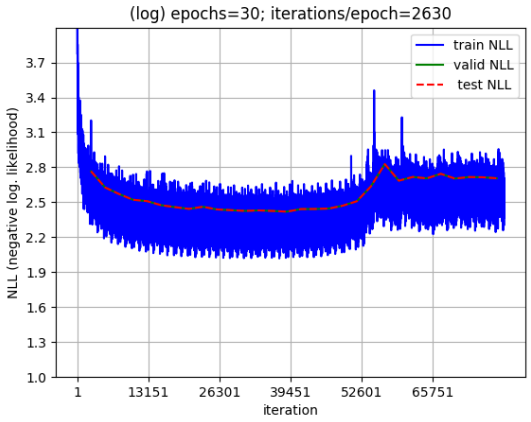


Figure 4.1: NLL loss curve using μ-law Quantification, Weight normalization and Adam optimizer

In short, the best results are achieved using the Scheduler. With RSMprop we get a clearly worse signal, but on the other hand the signals generated using or not the scheduler did not seem very different. When analyzing them more in detail, the synthesis when not using the Scheduler generated a clear signal with the occasional appearance of discontinuities. In contrary we obtained a clear signal when we use it. If we look at the figure 4.2, we can observe around 5.5 seconds a noise discontinuity, this is because the model can not correctly

Quantizer	Average NLL Test
$\mu-$law Quantitzation	**2.56**
Uniform Quantitzation	1.56

Table 4.5: The different quantifications used in the conditioned SampleRNN under Ahocoders output.

predict these samples. On the other hand, as shown in the figure 4.3, by using Scheduler we managed to predict these samples correctly.

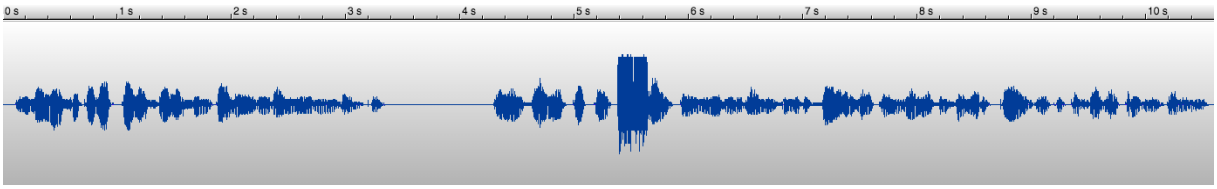


Figure 4.2: Audio wave generated with Adam optimizer.

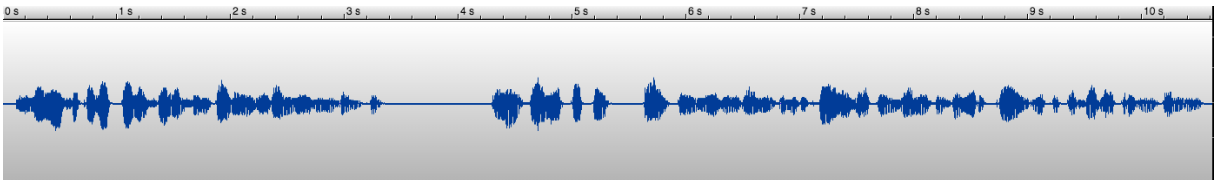


Figure 4.3: Audio wave generated with Adam optimizer using Scheduler method to adjust the learning rate.

Since in the analysis of the unconditional model we decide to use the μ - *law* quantification, because generated greater clarity in the random signal, we decided to re-evaluate both quantifications with the synthesized speech signal. Table 4.5 shows similar results than the obtained with the unconditional system, in terms of NLL, but definitely the signal generated with μ-*law* quantification offers more clarity than the uniform one.

4.2.3 TTS System: MUSA and SampleRNN

Once adapted SampleRNN to be conditioned we use MUSA to generate feature vectors directly from the text, and thus create the TTS complete-system. Initially, an evaluation with MUSA features vector on the previous best model, with the Ahocoder conditioning, was performed. As a result we obtained a distorted, but still intelligible signal.

Then, an implementation with both models trained separately was performed. First we trained MUSA until we do not improve the averaged loss in validation. Then with the best model we have obtained we generate the acoustic parameters and save them in separated files. Then we train SampleRNN with those files, just as we did with the Ahocoder files. Finally we propose a system that integrates both stages and we re-train it jointly.

4.2.3.1 MUSA and SampleRNN trained independently

As explained, we perform two experiments when training SampleRNN with the acoustic parameters generated by MUSA. First we train SampleRNN starting from scratch. In terms of averaged NLL we obtain a significantly worse result than with the acoustic characteristics extracted with Ahocoder. Also, as will be explained in the subjective analysis, we obtained a

clearly worse signal. Therefore we tried to train the system starting from the best Ahocoder-conditioned SampleRNN model.

Model	Average NLL Test
From Scratch	2.67
Pre-trained Sample	2.66

Table 4.6: Comparison of SampleRNN conditioned under MUSA from scratch vs starting from a pre-trained model with Ahocoder in terms of averaged NLL.

As shown in the Table we do not obtain a substantial improvement in the results. With the aim of improving them, we proposed implementing a new model that would simultaneously learn the generation of acoustic parameters, as well as the prediction of the sample.

4.2.3.2 MUSA and SampleRNN trained jointly

In this section the final system is implemented, in order to connect the output of the acoustic model with the input of SampleRNN. For this purpose, a new model composed of MUSAs acoustic model and SampleRNN vocoder is implemented. First we will load both pre-trained models, in order to not starting from scratch. Then each time a batch of inputs enters to the network, explained in the section 3.3, will crossbeam the acoustic model generating a vector of 13 normalized acoustic frames, this will condition the input and target vectors of the corresponding samples and enter to SampleRNN network. In this way, the optimization of both models is done in parallel, taking into account that we generate the most accurate acoustic frames to predict the most similar sample to the target. We achieved a significant improvement obtaining an averaged NLL in test of **2.47**, but even so the generated signal did not achieve the expected quality.

We knew that there would be a degradation of the signal, between the conditioned system with Ahocoder (which extracts the acoustic parameters from the signal) and the system conditioned by MUSA (which predicts the acoustic parameters from the text), but we did not expect such an exaggerated difference. Thus we conducted a study to compare those parameters.

First, we calculate the mean and variance of the normalized distributions of each parameter (MFCC, log(f0) and MVF) at the output of Ahocoder and MUSA.

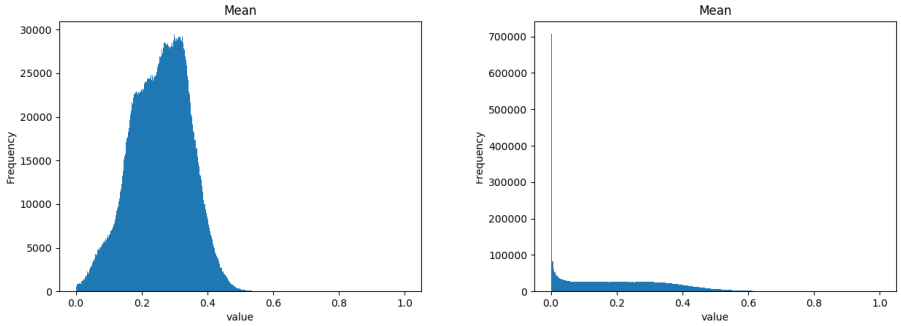
We can observe that there are no significant differences in the distributions of the mel cepstral coefficients, but instead the pitch contour as well as the Maximum voiced frequency have different means.

Finally we calculate the histograms of some cepstral coefficients, as well as the pitch and MVF at the output of both systems, which we enclose in the Appendices C. Generally, the histograms of each parameter are quite different. The most outstanding results that we obtain are the pitch distributions. As can be seen in the example below, there is a higher concentration around zero, which indicates a higher pitch than the original; also a flatter distribution is appreciated.

We can conclude that the problem lies in the fact that the prediction of the mentioned

	MFCC	log(f0)	MFV
MUSA	$\mu = 0.50$ $\sigma = 0.02$	$\mu = 0.18$ $\sigma = 0.02$	$\mu = 0.09$ $\sigma = 0.04$
Ahocoder	$\mu = 0.50$ $\sigma = 0.01$	$\mu = 0.25$ $\sigma = 0.00$	$\mu = 0.17$ $\sigma = 0.05$

Table 4.7: Mean and variance of the normalized distributions of MFCC, pitch and MVF.



(a) Logarithmic pitch distribution from Ahocoder

(b) Logarithmic pitch distribution from MUSA

Figure 4.4: Pitch histogram

parameters is not performed correctly. We believe that by jointly re-training the whole system, we slightly improve the prediction of these parameters, but without reaching similar results to those of Ahocoder.

4.3 Subjective results

Objective tests performed well to compare different architectures and to suggest how well does the TTS work. But when analyzing the complete system, that is, the conditioned SampleRNN under the MUSA characteristics, we obtained similar results in all the performed experiments. To get a more in-depth evaluation of the model a subjective test is conducted to evaluate the naturalness of the TTS developed in this work and also to make a comparison with the baseline system, that is, Ahodecoder vocoder under MUSA parametritzation.

In the next section we will explain the results obtained in a subjective study performed by a total of 14 volunteers who gave a rating to all the samples for each of the three experiments. A web based application was developed with the selected audio files obtained from the experiments, explained below, and volunteers were asked to rate the voices in a scale from one to five, one being bad and five being excellent. In the test each listener was asked to evaluate 5 sentences, randomly selected from the test set. The participants can listen the different recordings as many times as required to make comparisons between the different systems. For every sentence, the listeners evaluate 3 different versions generated with the following 3 different systems:

- SampleRNN conditioned under MUSA acoustic parameters, both systems trained indepently (Pipeline).
- SampleRNN conditioned under MUSA acoustic parameters and trained jointly. (Joint Estimation)
- Using Ahodecoder to generate speech from the acoustic parameters predicted by MUSA. (Baseline system proposed by Pascual de la Puente (2016))

Table 4.8 show the results of the subjective test. We can see that the results are good for the TTS developed in this work, as it gets the best values. We can note there that the joinly training of the purposed TTS system improves the perception of naturalness, thus confirming the objective measures 4.2.3.2. On the other hand, the system trained separately obtains very low results, being surpassed by the baseline system, Musa with Ahocoder. That is, the participants have mainly hesitated between the Baseline and the trained Jointly systems, discarding, in most cases, the Pipeline system.

Musa+SampleRNN (Joint Estimation)	MUSA (Baseline)	Musa+SampleRNN (Pipeline)	Baseline OR Joint Estimation	Pipeline OR Joint Estimation
36	15	3	10	6

Table 4.8: Subjective evaluation disaggregated results

These results are provisional, since we have not been able to solve problem with the prediction of the acoustic parameters generated by MUSA. In the next months we will continue with the work, in order to correct the detected error, with the aim of obtaining definitive results.

We have published the samples generated with the most relevant systems seen in the objective results, as well as the ones used for the subjective analysis in:

http://veu.talp.cat/tts/neural_vocoder/

The following chapter makes an estimation necessary budget to carry out this project. On the other hand, in the last chapter, the conclusions will be explained (6), where a review of the entire project will be made, discussing the implemented architectures and achieved results. Lines of future work are also explained.

Chapter 5

Budget

The main costs of this project comes from the salary of the researchers, and the server provided by the VEU Group of UPC.

With regard to the servers an estimatation of their hourly cost by using the prices of Amazon AWS EC2 servers Amazon Web Services has been made.

In relation to the salary that should be charged for the completion of the project it has been calculated a salary of Junior engineer for the total hours dedicated. Furthermore, for the professor, who were supervising and advising the project, we considered a reference salary of a senior engineer.

We will consider that the total duration of the project was of 27 weeks, as depicted in the Gantt diagram illustrated in the Annexes.

	Amount	**Wage/hour**	**Dedication**	**Total**
Junior engineer	1	15.00 €/h	40 h/week	16 200 €
Senior engineer	1	60.00€/h	4 h/week	6 480 €
	Amount	**Hours**	**Cost/hour**	**Total**
Servers	1	2352 h	0.99 €/h	2 328 €
			Total	25 008 €

Table 5.1: Estimated budget of the project

Chapter 6

Conclusions

In this project, a complete-system speech synthesizer has been implemented using deep learning. First MUSA predicts the duration of the phonemes to generate from the input text, and then the acoustic parameters to be passed to the Neural Vocoder are generated frame by frame up to the corresponding duration. Finally the Neural Vocoder, SampleRNN, will convert the parameterization of the speech, obtained from the text, into the respective waveform.

MUSA was a system that generated voice parameters, which were converted to voice by a vocoder. On the other hand, SampleRNN generated voice in an unconditional way. In this work we have modified SampleRNN to accept as a conditioner voice parameters. It has been proven on both parameters extracted directly from the speech as well as with the original predictions of MUSA. Finally, a system that includes both systems in a single neuronal network is implemented, allowing a joint training of the system, which takes the linguistic information from its input and generates speech samples. By optimizing the system as a whole the initial vocoder parameters are diluted by looking for a different representation that optimizes the metric used on the generated signal.

First of all, some variants have been introduced that have proven to be beneficial on the initial SampleRNN model. SampleRNN discretizes the signal with 8 bits to have a more flexible model of its pdf. The original work used a uniform quantization while here, influenced by the proposed Wavenet system (van den Oord et al., 2016), the law-mu quantification has been successfully proposed, significantly improving the quality of the generated signal. On the other hand we have added the normalization of weights in all the linear layers that make up the network, showing an improvement in the loss curves of the system. Finally we have carried out a study tuning the various parameters of the architecture. Through experimental results we have seen that it is preferable

As regards of "conditioning" SampleRNN, the unconditional model which generates audio with human resemblance but with no linguistic content, is extended. Specifically, the parameters extracted by Ahocoder: MFCC, lf0 and MVF are aded as an additional conditioner. This extension has been a success, obtaining voice of great quality and naturalness.

The next step has been the integration of the system with MUSA. In the first instance, direct integration was evaluated, obtaining results that differed greatly from the previous ones. Therefore, a re-estimation on SampleRNN was carried out with the parameters generated by MUSA independently, although without success, generating similar results. Finally, the joint re-estimation of both models, as a single network, turns out to be very beneficial, improving significantly the independent systems aquitecture results, but still not approaching the signal generated with the parameters extracted from the voice.

Unfortunately, in MUSAs estimation we have an error not yet detected, since the signal generated by MUSA with Ahodecoder is of poor quality. An analysis of the generated signal shows a histogram of $log(f0)$ which differs from a natural distribution, and with a higher value. In the coming months, we will continue with the work, investigating the detected error, with the aim of obtaining definitive results, and with the intention of submitting a paper to the International Conference InterSpeech'2018. However, the integration of the two systems and joint optimization has shown to be very beneficial since it has been able to correct the error in the pitch of the incorrect output of MUSA and significantly improve the quality of the signal. We perform a perceptual test, where 14 listeners evaluate 5 generated samples by the Baseline system and our system, trained separately and jointly, obtaining the best results our jointly trained system.

Furthermore, for future work, our SampleRNN reimplementation offers a Conditional Neural Vocoder that can be conditioned with other parameters. Therefore, a future line of work will be looking for alternative representations of the input conditioners, such as speaker and language identity in multispeaker and multilingua tts, the magnitude spectra, or the a neural front-end working directly with characters.

Once we rectify the error in the estimation of the acoustic parameters generated by MUSA, we will publish the code from this project, as a contribution to the scientific community. Also some synthesized examples from different architectures are available in http://veu.talp.cat/tts/neural_vocoder/.

Bibliography

Amazon Web Services. AWS | Amazon EC2 Dedicated Instances - Dedicated Hosting. https://aws.amazon.com/ec2/purchasing-options/dedicated-instances/. Accessed on 4-5-2017.

Sercan Ömer Arik, Mike Chrzanowski, Adam Coates, Greg Diamos, Andrew Gibiansky, Yongguo Kang, Xian Li, John Miller, Jonathan Raiman, Shubho Sengupta, and Mohammad Shoeybi. Deep voice: Real-time neural text-to-speech. *CoRR*, abs/1702.07825, 2017. URL http://arxiv.org/abs/1702.07825.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014. URL http://arxiv.org/abs/1409.0473.

Antonio Bonafonte, Pablo D Agüero, Jordi Adell, Javier Pérez, and Asunción Moreno. Ogmios: The upc text-to-speech synthesis system for spoken translation. In *TC-STAR Workshop on Speech-to-Speech Translation*, pages 199–204, 2006.

James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. Quasi-recurrent neural networks. *CoRR*, abs/1611.01576, 2016. URL http://arxiv.org/abs/1611.01576.

Denny Britz. Recurrent neural networks tutorial, 2017. URL http://ruder.io/deep-learning-optimization-2017/. [Online; accessed 17-January-2018].

Tim Capes, Paul Coles, Alistair Conkie, Ladan Golipour, Abie Hadjitarkhani, Qiong Hu, Nancy Huddleston, Melvyn Hunt, Jiangchuan Li, Matthias Neeracher, Kishore Prahallad, Tuomo Raitio, Ramya Rasipuram, Greg Townsend, Becci Williamson, David Winarsky, Zhizheng Wu, and Hepeng Zhang. Siri on-device deep learning-guided unit selection text-to-speech system. pages 4011–4015, 08 2017.

KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259, 2014. URL http://arxiv.org/abs/1409.1259.

TC-STAR ELRA-B0014. Tc-star spanish baseline male speech database. http://catalog.elra.info/product_info.php?products_id=1161.

Daniel Erro, Iñaki Sainz, Eva Navas, and Inma Hernáez. Improved HNM-Based Vocoder for Statistical Synthesizers. In *Interspeech 2011*, pages 1809–1812, Florence, Italy, August 2011. URL http://www.isca-speech.org/archive/interspeech_2011/i11_1809.html.

Daniel Erro Eslava. Ahocoder download - info. http://aholab.ehu.es/ahocoder/info.html, February 2017.

G. D. Forney. The viterbi algorithm. *Proc. of the IEEE*, 61:268 – 278, March 1973.

Pavel Golik, Patrick Doetsch, and Hermann Ney. Cross-entropy vs. squared error training: a theoretical and experimental comparison. In *INTERSPEECH*, 2013.

R. Hecht-Nielsen. Theory of the backpropagation neural network. In *International 1989 Joint Conference on Neural Networks*, pages 593–605 vol.1, 1989. doi: 10.1109/IJCNN.1989.118638.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. 9:1735–80, 12 1997.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, December 2014. URL http://arxiv.org/abs/1412.6980. arXiv: 1412.6980.

Y. LeCun, L. Bottou, G. Orr, and K. Muller. Efficient backprop. In G. Orr and Muller K., editors, *Neural Networks: Tricks of the trade*. Springer, 1998.

Soroush Mehri, Kundan Kumar, Ishaan Gulrajani, Rithesh Kumar, Shubham Jain, Jose Sotelo, Aaron C. Courville, and Yoshua Bengio. Samplernn: An unconditional end-to-end neural audio generation model. *CoRR*, abs/1612.07837, 2016. URL http://arxiv.org/abs/1612.07837.

Andrew Ng. Machine learning, Autum 2017. [Online; accessed 18-January-2018].

Michael A. Nielsen. Neural networks and deep learning. In *Determination Press*, December 2015. URL http://neuralnetworksanddeeplearning.com/.

Santiago Pascual de la Puente. Deep learning applied to speech synthesis. Master's thesis, Universitat Politècnica de Catalunya, 2016.

Sebastian Ruder. Optimization for deep learning highlights in 2017, September 2015. URL http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/.

Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *CoRR*, abs/1602.07868, 2016. URL http://arxiv.org/abs/1602.07868.

A. Schoenauer-Sebag, M. Schoenauer, and M. Sebag. Stochastic Gradient Descent: Going As Fast As Possible But Not Faster. *ArXiv e-prints*, September 2017.

Jose Sotelo, Soroush Mehri, Kundan Kumar, Joao Felipe Santos, Kyle Kastner, Aaron Courville, and Yoshua Bengio. Char2wav: End-to-end speech synthesis. In *International Conference on Learning Representations (ICLR): Workshop Track*, Toulon, France, April 2017.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. 15:1929–1958, 06 2014.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014. URL http://arxiv.org/abs/1409.3215.

Universitat Politècnica de Catalunya. Techonology and application of speech and language (talp). http://www.talp.upc.edu/. Accessed on 17-01-2018.

Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016. URL http://arxiv.org/abs/1609.03499.

Andrew J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, IT-13(2):260–269, April 1967.

Yuxuan Wang, R. J. Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron J. Weiss, Navdeep Jaitly, Zongheng Yang, Ying Xiao, Zhifeng Chen, Samy Bengio, Quoc V. Le, Yannis

Agiomyrgiannakis, Rob Clark, and Rif A. Saurous. Tacotron: A fully end-to-end text-to-speech synthesis model. *CoRR*, abs/1703.10135, 2017. URL http://arxiv.org/abs/1703.10135.

Wikipedia. Artificial neuron — Wikipedia, the free encyclopedia, 2005. URL https://commons.wikimedia.org/wiki/File:ArtificialNeuronModel_english.png. [Online; accessed 18-January-2018].

Takayoshi Yoshimura, Keiichi Tokuda, Takao Kobayashi, Takashi Masuko, and Tadashi Kitamura. Simultaneous modeling of spectrum, pitch and duration in hmm-based speech synthesis, 1999.

Heiga Zen, Keiichi Tokuda, and Alan W Black. Statistical parametric speech synthesis. *Speech Communication*, 51(11):1039–1064, 2009.

Heiga Zen, Andrew Senior, and Mike Schuster. Statistical parametric speech synthesis using deep neural networks. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 7962–7966, 2013.

Appendix A

Work Plan and Gantt Diagram

Project: Knowledges	**WP ref: WP1**	
Major constituent: theoretical learning	Sheet 1 of 6	
Short description: Study of generation of speech works and deep learning techniques.	Planned start date: 20/07/2017 Planned end date: 16/10/2017	
	Start event: Start Udacity Course End event: Finish Udacity Course	
Internal task T1: Deep Learning. (Udacity Course). Internal task T2: Waveform syntetizers. (Wavenet). Internal task T3: Character-based synthetizers. (Tacotron, Char2Wav, DeepVoice) Internal task T4: State of the art.	Deliverables:	Dates:

Project: Programming skills	**WP ref: WP2**	
Major constituent: programming learning	Sheet 2 of 6	
Short description: Learn to work with Deep Learning Python libraries.	Planned start date: 04/08/2017 Planned end date: 17/10/2017	
	Start event: Udacity Course End event: Pytorch knowledge acquired	
Internal task T1: Python. Internal task T2: Numpy. Internal task T3: TensorFlow. Internal task T4: Pytorch.	Deliverables:	Dates:

Project: SampleRNN Development (Unconditioned)	**WP ref: WP3**	
Major constituent: development of the neural vocoder of the system	Sheet 3 of 6	
Short description: Apply the obtained programming skills and the analysis of SampleRNN paper to understand and adapt the authors code.	Planned start date: 12/10/2017 Planned end date: 11/12/2017	
	Start event: SampleRNN code comprehension End event: SampleRNN evaluation / comparison	
Baseline task B1: SampleRNN code comprehension. Baseline task B2: Database preparation. Baseline task B3: System training. Baseline task B4: SampleRNN code adaptation: Mu law Quantizer, QRNN. Baseline task B5: System evaluation.	Deliverables: SampleRNN implementation.	Dates: 11/12/2017

Project: Conditioning SampleRNN with MUSA speech parametrization - Pipeline	**WP ref: WP4**	
Major constituent: Add conditoning to SampleRNN	Sheet 4 of 6	
Short description: Condition SampleRNN with MUSA speech parametrization.	Planned start date: 14/11/2017 Planned end date: 16/12/2017	
	Start event: SampleRNN code adaptation. End event: Finish pipeline system.	
Baseline task B1: Database preparation. Baseline task B2: SampleRNN adaptation for conditioning. Baseline task B3: System training. Baseline task B4: System evaluation.	Deliverables: Condicional SampleRNN implementation (Pipeline).	Dates 22/12/2017

<table>
<tr><td>Project: Integrate MUSA and neural vocoder</td><td colspan="2">WP ref: WP5</td></tr>
<tr><td>Major constituent: Integrate in a final system MUSA and neural vocoder</td><td colspan="2">Sheet 5 of 6</td></tr>
<tr><td rowspan="2">Short description: Train both systems: parametrization (MUSA) and vocoder (sampleRNN) in order to improve the pipeline system.</td><td colspan="2">Planned start date: 15/12/2016
Planned end date: 22/01/2018</td></tr>
<tr><td colspan="2">Start event: Pipeline system implemented/evaluated.
End event: Evaluate the final system.</td></tr>
<tr><td>Baseline task B1: Train SampleRNN with MUSA parametrization and vocoder samples.
Baseline task B2: Integrate MUSA and SampleRNN code.
Baseline task B3: Train MUSA and SampleRNN jointly.
Baseline task B4: Evaluate the new system.</td><td>Deliverables:
Systems evaluation report</td><td>Dates
18/01/2018</td></tr>
</table>

<table>
<tr><td>Project: Documentation</td><td colspan="2">WP ref: WP6</td></tr>
<tr><td>Major constituent: project reports deliveries</td><td colspan="2">Sheet 6 of 6</td></tr>
<tr><td rowspan="2">Short description: Write the different project reports.</td><td colspan="2">Planned start date: 19/09/2017
Planned end date:29/01/2018</td></tr>
<tr><td colspan="2">Start event: Proposal and Workplan Delivery
End event: Final Delivery.</td></tr>
<tr><td>Documentation task D1: Proposal and Workplan.
Documentation task D2: Project Critical Review.
Documentation task D3: Final Report.</td><td>Deliverables:
PW
PCR
FR</td><td>Dates
09/10/2017
01/12/2017
25/01/2018</td></tr>
</table>

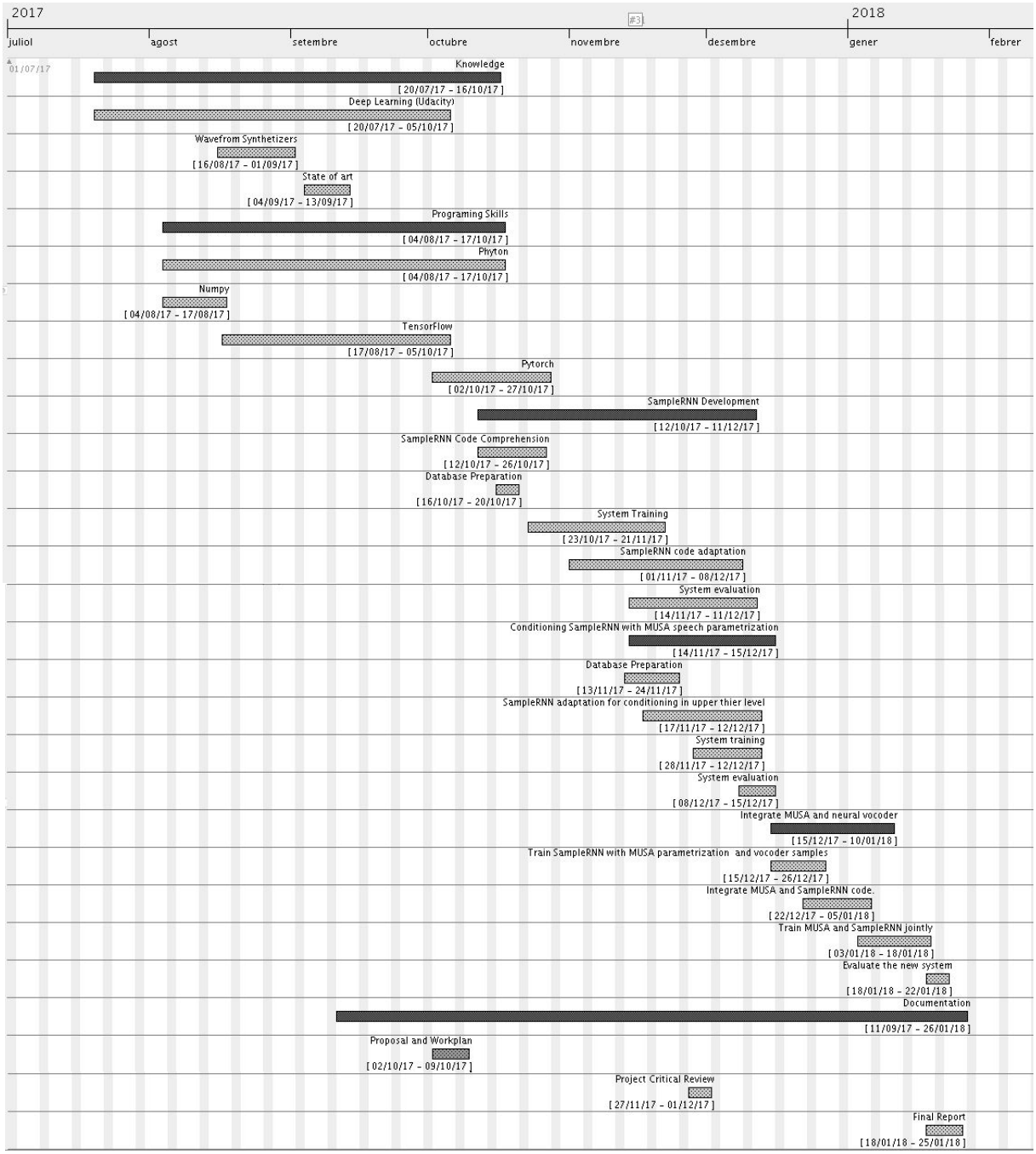
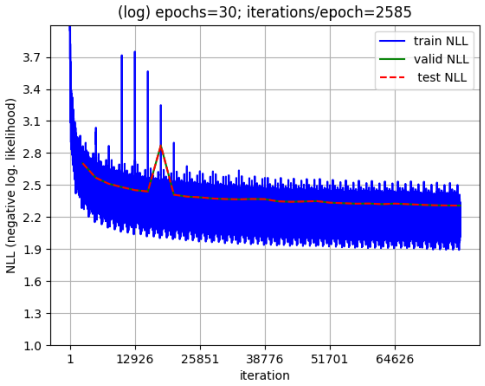


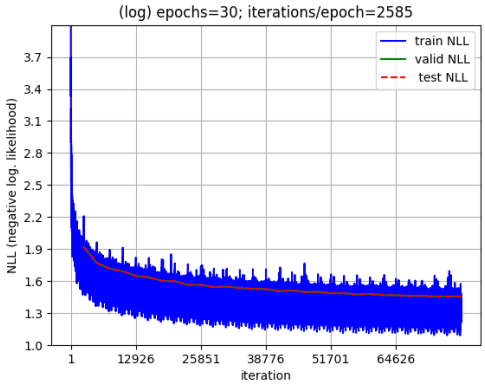
Figure A.1: Gantt Diagram

Appendix B

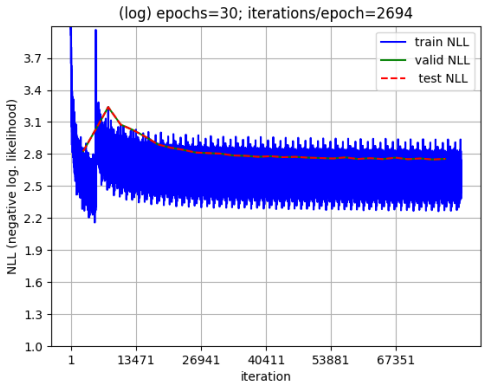
NLL Loss Curves



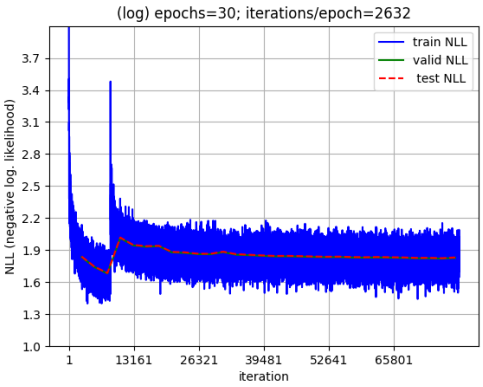
(a) Weight normalitzation and μ-*law* quantification



(b) Weight normalitzation and Uniform quantification



(c) μ-*law* quantification without Weight normalitzation



(d) Uniform quantification without Weight normalitzation

Figure B.1: Loss curves up to 30 epochs of all combinations between the use of weight normalization and the type of quantification.

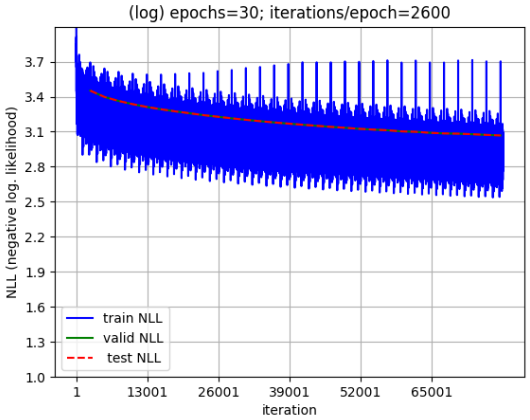


Figure B.2: NLL Loss curve when using RMSprop as optimizer.

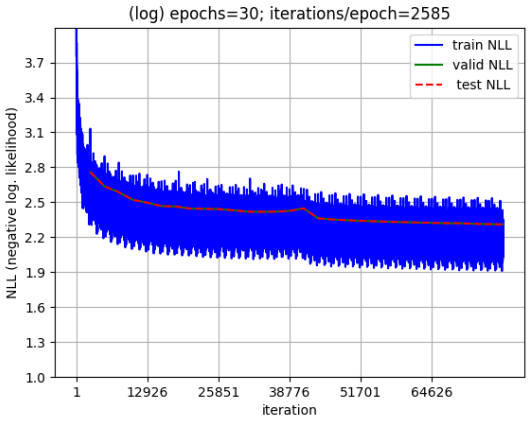
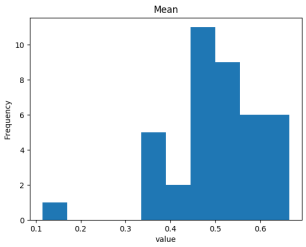


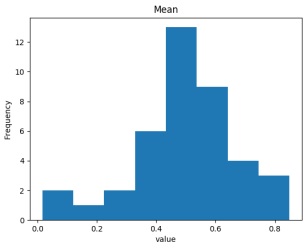
Figure B.3: NLL loss curve using μ-*law* Quantification, Weight normalization and Adam optimizer with Scheduler

Appendix C

Acoustic parameters histograms

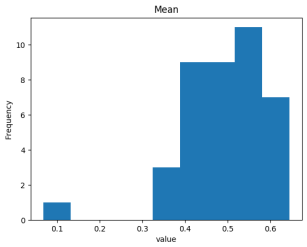


(a) Ahocoder

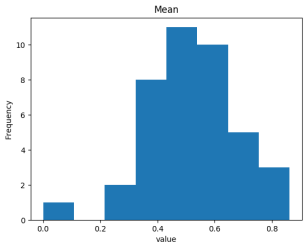


(b) MUSA

Figure C.1: Histogram of the Normalized 1th mel-cepstral coefficient

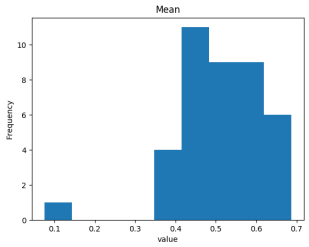


(a) Ahocoder

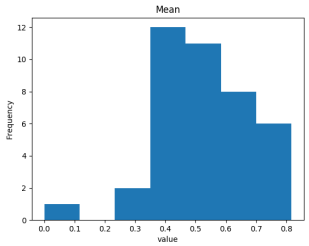


(b) MUSA

Figure C.2: Histogram of the Normalized 5th mel-cepstral coefficient

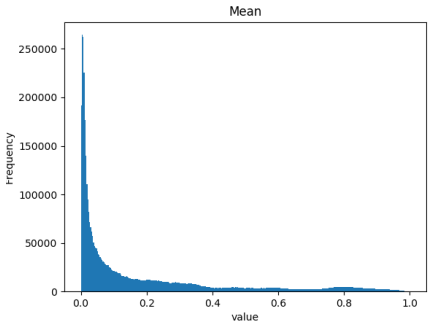


(a) Ahocoder

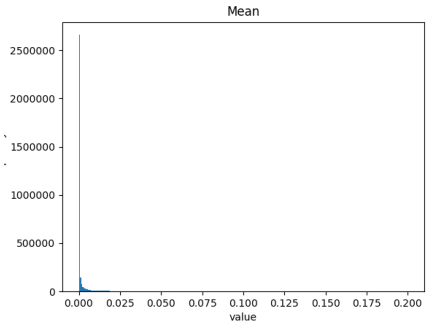


(b) MUSA

Figure C.3: Histogram of the Normalized 10th mel-cepstral coefficient

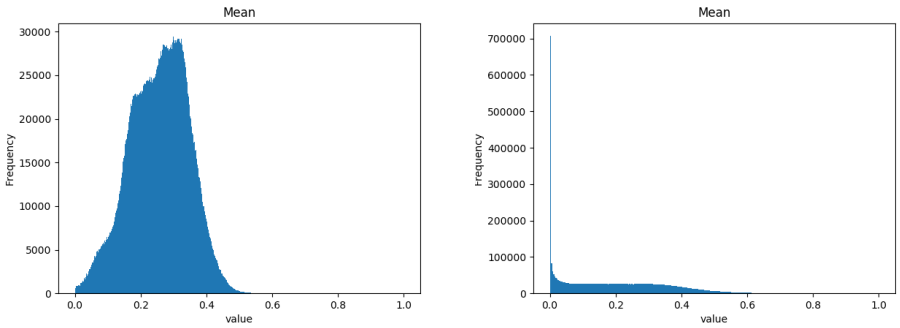


(a) Ahocoder



(b) MUSA

Figure C.4: Maximum Voiced Frequency histogram



(a) Ahocoder (b) MUSA

Figure C.5: Pitch histogram